1992

A Computational Framework for Efficient Error Correcting Codes Using an Artificial Neural Network Paradigm.

Maung Maung Htay
Louisiana State University and Agricultural & Mechanical College

Order Number 9316987

A computational framework for efficient error correcting codes using an artificial neural network paradigm

Htay, Maung Maung, Ph.D.

The Louisiana State University and Agricultural and Mechanical Col., 1992

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

A COMPUTATIONAL FRAMEWORK FOR EFFICIENT ERROR CORRECTING CODES USING AN ARTIFICIAL NEURAL NETWORK PARADIGM

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirement for the degree of
Doctor of Philosophy

in

The Department of Computer Science

by
Maung Maung Htay
B.S., University of Rangoon, Burma, 1971
M.S., University of London, United Kingdom, 1976
M.S., University of Rangoon, Burma, 1979
December, 1992

Acknowledgements

This dissertation would not have been possible without the encouragement and help of numerous people. First, I sincerely express my gratitude and appreciation to Professor S. Sitharama Iyengar, for the invaluable advice, motivation, encouragement and guidance that he provided throughout my stay at Louisiana State University.

Second, I would like to thank my committee members: Professors Donald Kraft, Bush Jones, Doris Carver, Gisele Goldstein, and Bogdan Oporowski for their encouragement, support and constructive comments. My discussions with Professors Bogdan Oporowski and Si-Qing Zheng were extremely fruitful, and gave me some insight to the research problems that are discussed here. I wish to express my profound gratitude to all my teachers, past and present, especially the faculty members in our department, for providing me with a strong foundation to build on the work described here. My friends especially Weian Deng, Fenglien Lee and Jigang Liu made my stay pleasant. U Hla Min read and edited this dissertation.

Financial support was provided by the United States Information Agency (USIA) in the form of a Fulbright Scholarship. I appreciate the services of the USIA officers and the people back at home who selected me to study in the United States.

Special thanks go to members of my family: my mother, Daw Mya Nyunt, my sister, Daw Pyone Kyi, my lovely wife, San Yin, my precious son, Aung Ko Ko, and my two beautiful daughters, Lin Lin Maung and Aye Su Mon, for their constant support, love, generosity, patience and sacrifice. They shared my pains, and without their moral support, I would not have survived the lonely days at LSU, far from my family and home, to complete my work. I will always be grateful to them.

Last but not the least, I wish to dedicate this dissertation to my beloved father U Ba Tin, who passed away in 1953, for nurturing my confidence and hardwork.

Table of Contents

List of Tables

List of Figures

Abstract

The quest for an efficient computational approach to neural connectivity problems has undergone a significant evolution in the last few years. The current best systems are far from equaling human performance, especially when a program of instructions is executed sequentially as in a von Neuman computer. On the other hand, neural net models are potential candidates for parallel processing since they explore many competing hypotheses simultaneously using massively parallel nets composed of many computational elements connected by links with variable weights. Thus, the application of modeling of a neural network must be complemented by deep insight into how to embed algorithms for an error correcting paradigm in order to gain the advantage of parallel computation.

In this dissertation, we construct a neural network for single error detection and correction in linear codes. Then we present an error-detecting paradigm in the framework of neural networks. We consider the problem of error detection of systematic unidirectional codes which is assumed to have double or triple errors. The generalization of network construction for the error-detecting codes is discussed with a heuristic algorithm. We also describe models of the code construction, detection and correction of t-EC/d-ED/AUED (t-Error Correcting /d-Error Detecting/All Unidirectional Error Detecting) codes which are more general codes in the error correcting paradigm.

Chapter 1

General Introduction

1.1 Overview

Reliable information is an asset in communications, control and computing. It forms the basis of efficient data processing, information processing, knowledge processing and intelligence processing. On the other hand, wrong or corrupt information can cause confusion, misunderstanding, hatred, disasters and wars. It is thus desirable to maintain the integrity of information in all phases of computation. Despite advances in hardware for computer and data communications, errors will invariably occur as information is being stored, transferred or manipulated and it is necessary for such systems to incorporate automatic error detection/correction[8]. Error detecting/correcting codes have been extensively discussed for improving the reliability of computer systems and communication networks. A lot of publications have been written about error detecting/correcting codes. Among them, Richard Hamming pioneered a form of coding amenable for error detection and correction. For example, a SEC(Single Error Correction) Hamming code is generally sufficient for computer systems, where the probability of multiple-bit error is low.

Some factors that can contribute to errors in communication are atmospherics, electrical noise, component failures, device malfunctions and design faults[31]. Error detection enable reliable information processing. Error correction/recovery can prevent serious system crashes and allow fault-tolerant computing.

Numerous algorithms have been developed for error detecting/correcting codes. Some have been applied in the design of fault-tolerant computers which use ROM,

LSI, VLSI, RAM and PLA[5][6][24][25][30]. To enhance reliance, built-in error detecting/correcting features have been incorporated in megabit-level chip designs. On-the-chip codes have also been developed to improve yield and enhance testability of semiconductor chips[31].

There are several approaches in the research on error correcting codes. Error may be symmetric, random or unidirectional. Algorithms for each class of the error have been developed quite successfully. The major problem is how to handle a general class of errors encompassing one or more types.

Some notable work in this area has been reported. B. Bose and Rao [5] have developed a theory on unidirectional error correcting/detecting codes. D. Nikolos[25] has presented the fundamental theory of t-error correcting/d-error detecting ($d > t$) and all unidirectional error detecting (t-EC/d-ED/AUED). D. Nikolos, N. Gaitanis and G. Philokyprou[24] have also given different methods for t-random error correcting and all unidirectional error detecting codes. B. Bose and D. K. Pradhan[6] have shown that their t-error correcting and multiple unidirectional error detecting systematics codes are more efficient than the earlier codes. Also B. Bose[7] has described a parallel unordered coding scheme with 2^r information bits and r check bits as an extension of Knuth's results [20]. B. Bose and D.J. Lin [8] have worked on systematic unidirectional error-detecting codes.

1.2 Scope of the Dissertation

In this dissertation, we attempt to address a general problem of error correcting codes in a computational framework of the neural network paradigm. We use some work described above to develop new algorithms using a neural network. Since the

brain consists of neurons of different types which cooperate and coordinate to carry out tests that present day computers are still struggling to perform. For instance, "a year old baby is much better and faster at recognizing objects, faces, and so on than even the most advanced AI systems running on the fastest supercomputer"[14].

Neural computing has an important role in structuring a biological system. Inexpensive, powerful micro computers, for instance, can be combined to form an MPP (Massively Parallel Processor) that is capable of implementing good and accurate models of biological systems. Biological systems are subject to physiological complexities and it is often difficult to predict behavior during experimental investigation.

There are different facets of neural computing. We will concentrate on the construction of neural nets to design and implement error detecting and/or correcting codes for some classes of error types. We will develop on error correcting paradigm for use in neural networks.

1.3 Organization of the Dissertation

This dissertation is organized as follow. In Chapter 2, the basic concept of codes including classification of codes, fundamental perception of neural networks, inspiration from neuroscience and its basic model such as the McCulloch and Pitts model are given as preliminaries. Single error correcting in linear codes using neural network are discussed in Chapter 3. Since unidirectional errors appear in many cases, we describe systematic unidirectional error-detecting codes with the use of neural network in Chapter 4. We also present neural network construction in t-error correcting/d-error detecting (d>t) and all unidirectional error detecting codes in Chapter 5. Then we summarize and discuss future work in this research area in Chapter 6.

Chapter 2

Preliminaries

2.1 Introduction

In this chapter, we present preliminary information of error-correcting codes and the basic concepts of neural networks. We describe the fundamental concept of codes in Section 2.2 and the framework of neural computing in Section 2.3.

2.2 Information Code Structures

The subject of error-correcting codes has been an established field of study for over three decades and many applications have accelerated in communication. In recent years, many types of error-correcting codes have been applied to computer systems such as memory system, and arithmetic processors. In communication, message sending and message receiving are major concerns. When messages are transmitted over a long distance, there may be some interference and the messages may not be received exactly as it is sent. Even over a short distance, incorrect messages may be received due to unreliable devices or a bad communication channel. Under these circumstances we need to detect and, if possible, correct errors. Thus many researchers have tried to find ways of constructing reliable messages. A systematic approach is to represent messages by selected words in a binary alphabet $\{0,1\}$, since every error is simply the result of confusing 0 and 1. The constructed binary word corresponding to a message must be satisfied to an agreed set of rules, known to both the sender and

receiver. The flow diagram for transmitting a message is shown in Figure 2.1.

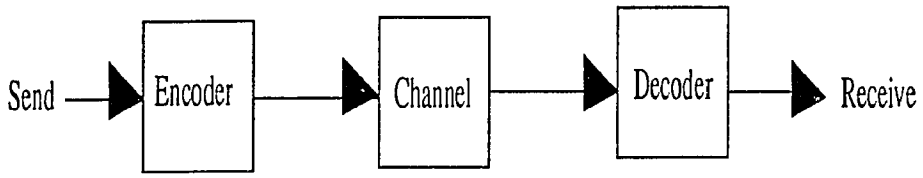

Figure 2.1 The Communication Channel

Formally a set of constructed binary words is called a *code* and the members of this set are called *codewords*. Generally codewords are formed by concatenating information bits and parity check bits which are computed by a formula using information bits.

2.2.1 Classes of Codes

There are two kinds of codes, *separable* and *nonseparable* codes. It is important to note that separable codes have significant advantages over non-separable codes. In separable codes, the information bits are separately identified from the check bits. Separable codes are also called as *systematic* codes. Even though certain applications have been involved with nonseparable codes [30][31], these codes still have only a very limited use. For instance, encoding items such as addresses, operands, and certain microinstructions use only separable codes[28][30].

Separable error-correcting/detecting codes are effective against transient, intermittent, and permanent faults. Transient faults are likely to cause a limited number of symmetric errors or multiple unidirectional errors whereas permanent faults cause either symmetric or unidirectional errors depending on the nature of the faults

[6][25][30]. Types of errors are formally defied in Section 4.2.1. Intermittent faults, because of short duration, are expected to cause a limited number of errors. In recently developed memories such as LSI/VLSI, ROM and RAM, the most likely faults cause unidirectional errors [24][25]. The number of symmetric errors is usually limited while the number of unidirectional errors caused by the above mentioned faults can be fairly large. Thus we can count on the fact that error correcting is the best way to cope with transient and intermittent faults [24][25].

Since transient faults are likely to be caused by random errors and unidirectional errors, it is better to consider a combination of random error correction along with unidirectional error detection and correction. Thus based on this concept, error correcting and detecting codes can be classified as below[30]:

C1: Detection of all patterns of t or few random errors and all unidirectional errors.

C2: Detection of all error patterns that consists of a combination of t or fewer random errors along with any number of unidirectional errors.

C3: Correction of all patterns of t or fewer random errors, and detection of d unidirectional errors, where $d \geq t + 1$.

C4: Correction of t or fewer random errors and detection of d errors, $d \geq t + 1$, containing at most t random errors.

According to the above classes, we will consider how to construct neural networks for correcting/detecting the following codes which are mostly useful in practice.

• Single error detecting and correcting linear codes.

• Systematic unidirectional error-detecting codes.

• t-error correcting/d-error detecting (d > t) and all unidirectional error detecting codes.

2.3 Neural Networks

The study of neural networks has grown rapidly during the past decade. The adjective "neural" is used primarily because much of the inspiration for such networks came from neuroscience, and not because of their relation to networks of real neurons [14]. In this research, the term *neural network*, is intended to mean an artificial neural network (ANN). Artificial neural network models or simply *neural nets* can be further classified as connectionist models, parallel distributed processing models, and neuromorphic systems. Although the names vary, the underlying concept of all these models is the same, that is to achieve good performance via dense interconnection of simple computational elements. In general, neural nets have been studied for many years for the purpose of achieving human-like performance in the fields of speech and image recognition. These models are composed of many nonlinear computational elements operating in parallel and arranged in patterns reminiscent of biological neural nets.

Even the current best systems are far from equaling human performance especially when a program of instructions is executed sequentially as in a von Neuman computer. On the other hand, neural nets models are potential candidates for parallel processing since they explore many competing hypotheses simultaneously using massively parallel nets composed of many computational elements connected by links with variable weights. Basically computational elements or nodes used in neural nets are nonlinear and typically analog [22]. The simplest neural net model sums N

weighted inputs and passes the result through a nonlinearity as shown in Figure 2.3. Three common types of nonlinearities are also shown in Figure 2.4.

Research work on artificial neural nets has a long history. Researchers believe that development of detailed mathematical models began more than 40 years ago with the works of McCulloch and Pitts [29], Hebb [13], Rosenblatt [33], Widrow [34], and others [5]. The work of Hopfield [15][16][17], Rumelhart and McCleland [9], Seijnowski [26], Feldman [11], Grossberg [12], and others have led to a new resurgence of this field very recently [22]. This new interest is due to the development of new analog VLSI implementations techniques [3], and some intriguing demonstrations [17][26] as well as by a growing fascination with the functioning of the human brain. Neural nets provide one technique for obtaining the required processing capacity by using large numbers of simple processing elements operating in parallel [22].

2.3.1 Inspiration from Neuroscience

The motivation of neural computation depends on the possibility of making artificial computing networks. A primary objective of a neural network is aimed more towards modeling networks of real neurons in the brain since the human brain is superior to a digital computer at many tasks. A good example is the processing of visual information. As we said before a human is much better than the most advanced super computer system in recognizing objects and faces, no AI systems running on the most powerful computer can beat even a year old baby in vision.

The brain is composed of about 10^{11} *neurons* of many different types. A schematic drawing of a typical neuron is shown in Figure 2.2. In this figure, a tree-like networks of nerve fibers called *dendrites* are connected to the *cell body* or *soma*,

where the cell nucleus is located. The *axon*, a single long fiber, is an extension of the cell body. There are branches or *arborizes* with strands and substrands in each axon. At the ends of these are transmitting ends of the *synaptic junctions*, or *synapses* to other neurons. The receiving ends of these junctions on other cells can be found both on the dendrites and on the cell bodies themselves. A few thousand of synapses are connected among neurons.

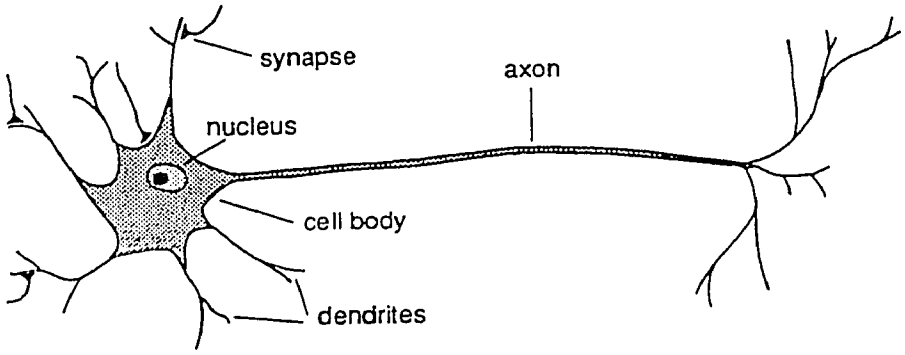

Figure 2.2 Schematic Drawing of a Typical Neuron

The firing process of a cell is the transmission of a signal from one cell to another at a synapse. The receiving cell accepts the electrical potential inside its body. If the potential reaches a threshold, a pulse or *action potential* of fixed strength and duration is sent down the axon. From this concept, McCulloch and Pitts [1943] propose a simple model of a neuron known as a binary threshold unit.

2.3.2 McCulloch and Pitts Model

McCulloch and Pitts's model computes a weighted sum of its inputs from other units and outputs a one or zero according to whether the sum is above or below a certain threshold. Figure 2.3 describes a simple model of a neural net[14].

In Figure 2.3, the weight w_{ij} represents the strength of the synapse connecting neuron i to neuron j and it can be positive or negative depending on whether synapse is excitatory or inhibitory. If there is no synapse between neuron i and neuron j, it is zero. μ_i represents the threshold.

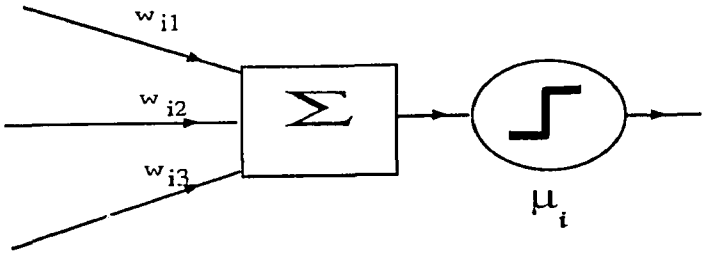

Figure 2.3. Schematic diagram of a McCulloch-Pitts neuron.

Mathematically [14], we can express the concept of the above model as:

$$\eta_i(t+1) = \Theta(\sum_j w_{ij}\eta_j(t) - \mu_i) \qquad (1)$$

where η_i is either 1 or 0 and represents the state of neuron i as firing or not firing at a given time respectively. Time t is taken as discrete, with one time unit elapsing per processing step. Θ (x) is the unit step function. This function is also known as threshold function. All neurons may not have the same fixed delay ($t \Rightarrow t + 1$). They are not updated synchronously by a central clock. Thus a simple generalization of the McCulloch-Pitts equation (1), which includes asynchronous updating and some other features such as continuous-valued units is

$$\eta_i = g(\sum_j w_{ij}\eta_j - \mu_i) \qquad (2)$$

where η_i is continuous-valued and is called the state or activation of unit i, g(x) is

In Figure 2.3, the weight w_{ij} represents the strength of the synapse connecting neuron i to neuron j and it can be positive or negative depending on whether synapse is excitatory or inhibitory. If there is no synapse between neuron i and neuron j, it is zero. μ_i represents the threshold.

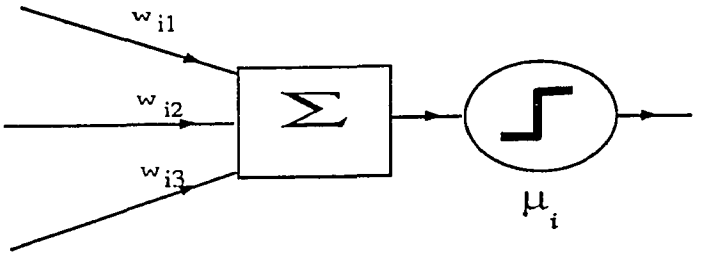

Figure 2.3 Schematic Diagram of a McCulloch-Pitts Neuron

Mathematically [14], we can express the concept of the above model as:

$$\eta_i(t+1) = \Theta(\sum_j w_{ij}\eta_j(t) - \mu_i) \qquad (1)$$

where η_i is either 1 or 0 and represents the state of neuron i as firing or not firing at a given time respectively. Time t is taken as discrete, with one time unit elapsing per processing step. Θ (x) is the unit step function. This function is also known as threshold function. All neurons may not have the same fixed delay ($t \Rightarrow t + 1$). They are not updated synchronously by a central clock. Thus a simple generalization of the McCulloch-Pitts equation (1), which includes asynchronous updating and some other features such as continuous-valued units is

$$\eta_i = g(\sum_j w_{ij}\eta_j - \mu_i) \qquad (2)$$

where η_i is continuous-valued and is called the state or activation of unit i, g(x) is

may be different for different i, and activation function $g(x)$ can also be made as side-dependent. We can assume that these weights and activation functions are stored by the processors as local data.

Even though there are many terms in the weighted sum in (2), the high connectivity of the network means that errors in a few terms will probably be inconsequential [14]. Thus such a system can be expected to be robust and its performance will degrade gracefully in the presence of noise or errors. Nerve cells in the brain itself die every day without affecting its performance significantly, and this robustness of the biological neural networks has probably been essential to the evolution of intelligence.

Neural nets models have many processors, each executing a very simple program, instead of the conventional situation where one or just a few processors execute very complicated programs. In contrast to the robustness of a neural network, an ordinary sequential computation may easily be ruined by a single bit error. "Nevertheless, the brain can do very fast processing for tasks like vision, motor control, and decisions on the basis of incomplete and noisy data, task that are far beyond the capacity of a current supercomputer" [14]. It seems possible only because billions of neurons operate simultaneously.

2.3.4 Real Time Implementation

At the present time, almost everything in the field of neural computation has been done by simulating the networks on serial computers, or by theoretical analysis. "Hardware construction of neural network VLSI chips are likely far behind the models since one needs a lot of connections, often some fraction of the square of the number of units. The connection space is a major factor in limiting the size of a network. So

far, a typical neural chip contains the order of 100 units but most practical applications need much dense units"[14]. To take full advantage of the capabilities of neural networks, it is necessary to design and build efficient hardware for handling ANNs.

Chapter 3

Correcting Errors in Linear Codes

3.1. Introduction

In this chapter, we present first some definitions and a theorem related to linear codes and then describe the construction of a neural net for error detection and correction with an illustrative example.

3.2 Linear Codes

In this section, we state some definitions and a theorem from [1][23] as background information concerning our proposed model for detecting and correcting errors in linear codes within the the framework of neural networks.

Let V^n be the set of all binary words of length n. A binary *code* of length n is simply a subset C of V^n and the members of C are called *codewords*.

Definition 3.1 : A code C in V^n is linear if whenever $a, b \in C$ then $a + b \in C$. In other words, C is linear *iff* it is a subgroup of V^n in Z_2, where Z_2 is the set of integers modulo 2. According to Lagrange's theorem, since a linear code is a subgroup of V^n, its size $|C|$ is a divisor of $|V^n| = 2^n$. Hence $|C|$ is an integer of the form 2^k, $0 \leq k \leq n$ and k is called the dimension of C.

We can also define the linear code in terms of a parity-check matrix as follows.

Definition 3.2 : A code C is a linear code if it is defined by $C = \{x \in V^n | H\ x' = 0'\}$ where H is a binary matrix with n columns and is known as parity-check matrix (or simply, check matrix), x' denotes the word x in V^n considered as a column vector and $0'$ denotes the all-zero column vector.

For a detailed knowledge of linear codes, the reader may refer to references [1][23]. Here, we will present some definitions and a theorem relevant to our problem.

Theorem 3.1 : If no column of H consists entirely of zeros, and no two columns are the same, then the code C defined by the check matrix H will correct one error.

The proof is given in reference [1].

A conventional algorithm [1] for detecting and correcting a single error in the code C is as follows:

begin

Let z be the received word.

Compute Hz' (z' is transpose of z)

if $Hz' = 0'$ **then** z is a codeword

else begin

Find the column $h^{(i)}$ of H

such that $H\ z' = h^{(i)}$

ith bit of z is incorrect.

Complement the *ith* bit of z

end

end

In the following section, we construct our proposed network model of correcting errors in linear codes assuming that all the necessary conditions prescribed in the above definitions and theorem are satisfied.

3.3 Neural Network Construction

To solve the given problem, we need to construct a neural net of two phases. In the first phase, the net detects an error position in the codeword. In the second phase, the net corrects the erroneous bit and produces the correct codeword.

3.3.1 Error Detecting Phase

It is shown in Figure 3.1. There are N inputs to the network, which consists of associative memories comprising of two layers of neurons. The number N represents the length of a codeword. In the hidden layer (also referred to as layer 1), there are M neurons where M represents the number of rows in the check matrix H which is assumed to be given. Codewords are defined depending on the check matrix. The reader may refer to the references [1][23] for more knowledge of the check matrix. Every input is connected with each neuron of layer 1. Layer 2 has N neurons which determine the position of the error of the received word. Every neuron of layer 1 is connected to each neuron of layer 2. Neurons in every layer are numbered by positive integer in consecutive increasing order starting from 1.

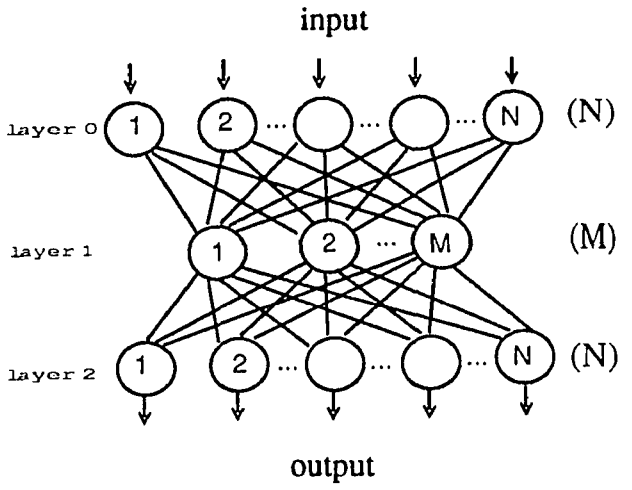

Figure 3.1 Error Detecting Phase (First Phase)

The elements in the given check matrix are used as weights at the connections of the N input with the nodes of the first layer. Let H be the given check matrix and h_{ij} the element at *ith* row and *jth* column of H.

We denote the weight w_{ij}^{01} of the connection of the *ith* neuron of the input layer with the *jth* neuron of layer 1.

$$w_{ij}^{01} = h_{ji} \text{ , } 1 \le i \le N \text{ and } 1 \le j \le M$$

We use w_{ij}^{12} to denote the weight of the connection of the *ith* neuron of layer 1 with the *jth* neuron of layer 2. The values of w_{ij}^{12} are also assigned by the elements of the check matrix H, but in the bipolar form (digit 0 is replaced by -1),

i.e. for $1 \le i \le M$, $1 \le j \le N$.

$$w_{ij}^{12} = \begin{cases} 1 & \text{if } h_{ij} = 1, \\ -1 & \text{if } h_{ij} = 0, \end{cases}$$

For each neuron of layer 1, the sgn function [Figure 3.2a] of the modulo 2 function is used as the activation function, while the hard limiter activation function [Figure 3.2b] used for the neurons of layer 2 [10][22].

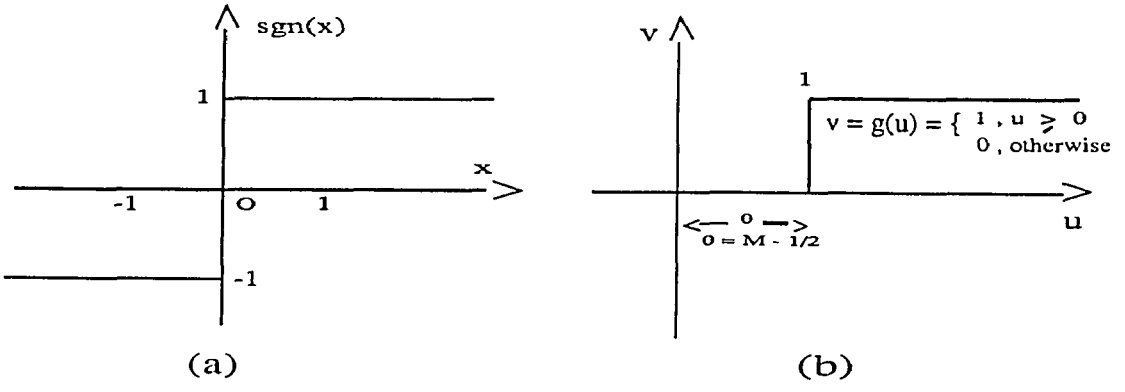

Figure 3.2 Activation Functions Used in the Proposed Network

To detect the error, the received word is passed through the first layer and we allow the network to progress until it falls into a stable situation. In this case, if one neuron of layer 2 produces value '1' while the rest are '0', then it shows the position of the bit which has been transmitted incorrectly. If there is no error in the received word, all neurons of layer 2 will output 0's.

To demonstrate our solution for detecting an error, we introduce the following variables and activation functions.

(1) The initial input v_j^0, $1 \le j \le N$

(2) The output of neuron t in layer 1 v_t^1, $1 \le t \le M$

(3) The output of neuron i in layer 2 v_i^2, $1 \le i \le N$

Let g^1 and g^2 be the activation functions for neurons of layer 1 and layer 2 respectively. g^1 is a sign function of modulo 2 function on the weighted sum of given inputs v_j^0, where $1 \le j \le N$. We denote the sign function as *sgn*.

Let $u_t^1 = \sum_j^N w_{jt}^{01} v_j^0$, $1 \le t \le M$ and $v_t^1 = g^1(u_t^1) = sgn(u_t^1 \, modulo 2)$,

$$i.e. \quad v_t^1 = g^1(u_t^1) = \begin{cases} 1, & \text{if } u_t^1 \bmod 2 = 1 \\ -1, & otherwise \end{cases} \qquad (3.1)$$

The output values of the neurons of layer 2 are determined by a hard limiter function g^2 [10][22].

Let $u_i^2 = \sum_j^M w_{ji}^{12} v_j^1$, $1 \le i \le N$ and $\theta = M - 1/2$, then we have,

$$i.e. \quad v_i^2 = g^2(u_i^2) = \begin{cases} 1, & \text{if } u_i^2 \ge \theta \\ 0, & otherwise \end{cases} \qquad (3.2)$$

In other words, since the *ith* neuron of layer 2 accepts as input the value u_i^2, where $u_i^2 = M$ and $u_i^2 > \theta$, the output v_i^2 will be equal to 1. For each neuron $j \ne i$ of layer 2, it holds that $u_j^2 < M - 1$ and $u_j^2 < \theta$. Therefore the output v_j^2 will be equal to 0 [10].

3.3.2 Error Correcting Phase

In this second phase, we use the exclusive or (XOR) network for finding the correct codeword. There are many methods for constructing the XOR network. In this case, we adopted two methods from [14] which are shown in Figures 3.3a and 3.3b.

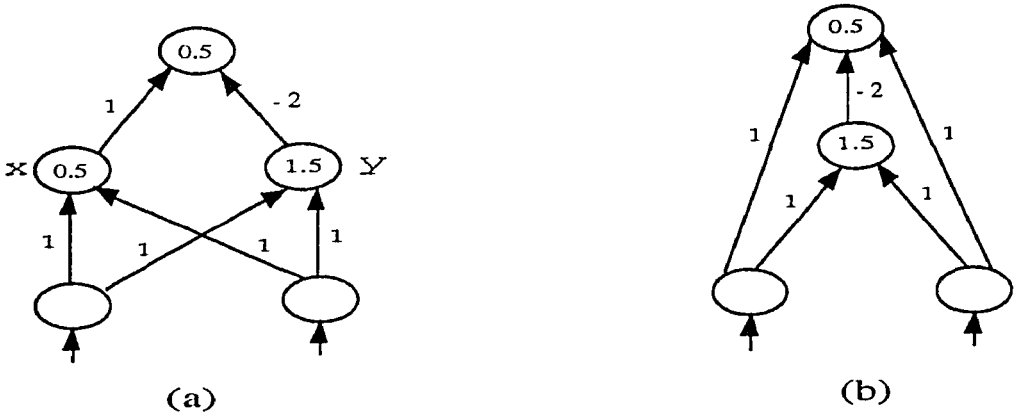

Figure 3.3 Networks for XOR

These networks use 0/1 threshold units. Each unit is shown with its threshold. In Figure 3.3a, the two neurons in the hidden layer compute the logical OR (left neuron x) and AND (right neuron y) of the two inputs and the output fires only when the x neuron fires and the y neuron does not fire.

The second method [Figure 3.3b] needs only two neurons, and one in the hidden layer computes a logical AND to inhibit the output unit when both inputs are on [14].

We use the corresponding pairs of bits from the output of phase 1 and the received word as the input to one of the XOR networks shown in Figure 3.3. The output of the second phase will be the correct codeword that we have expected.

3.4 An Illustrative Example

In this section, we use an example to demonstrate how error detection and error correction work using our proposed network.

Let C be the linear code defined by the check matrix

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

If the word 110110 is received and only one error has occurred, we can find the intended codeword by using our proposed network.

In this problem, since the number of rows in the check matrix is 3 and the length of the codeword is 6, we have M = 3 and N = 6.

3.4.1 Error Detection

The weights of the synapse connecting between input layer 0 and layer 1 are:

$$w_{11}^{01} = 1, \quad w_{12}^{01} = 1, \quad w_{13}^{01} = 1$$
$$w_{21}^{01} = 1, \quad w_{22}^{01} = 1, \quad w_{23}^{01} = 0$$
$$w_{31}^{01} = 0, \quad w_{32}^{01} = 0, \quad w_{33}^{01} = 1$$
$$w_{41}^{01} = 1, \quad w_{42}^{01} = 0, \quad w_{43}^{01} = 1$$
$$w_{51}^{01} = 0, \quad w_{52}^{01} = 1, \quad w_{53}^{01} = 0$$
$$w_{61}^{01} = 1, \quad w_{62}^{01} = 0, \quad w_{63}^{01} = 0$$

Inputs for the layer 1, i.e., bits of the word received, are

$$v_1^0 = 1,\ v_2^0 = 1,\ v_3^0 = 0\ v_4^0 = 1,\ v_5^0 = 1,\ v_6^0 = 0$$

According to the proposed network, we need to find the weighted sum of these inputs as follows:

$$u_1^1 = \sum_j^N w_{j1}^{01} v_j^0 = 1.1 + 1.1 + 0.0 + 1.1 + 0.1 + 0.0 = 3$$

$$u_2^1 = \sum_j^N w_{j2}^{01} v_j^0 = 1.1 + 1.1 + 0.0 + 0.1 + 1.1 + 0.0 = 3$$

$$u_3^1 = \sum_j^N w_{j3}^{01} v_j^0 = 1.1 + 0.0 + 1.0 + 1.1 + 0.1 + 0.0 = 2$$

The outputs of neurons in the layer 1 are :

$$v_1^1 = g^1(u_1^1) = 1$$
$$v_2^1 = g^1(u_2^1) = 1$$
$$v_3^1 = g^1(u_3^1) = -1$$

The weights of the synapse connecting layer 1 and layer 2 are :

$$w_{11}^{12} = 1, \quad w_{21}^{12} = 1, \quad w_{31}^{12} = 1$$
$$w_{12}^{12} = 1, \quad w_{22}^{12} = 1, \quad w_{32}^{12} = -1$$
$$w_{13}^{12} = -1, \quad w_{23}^{12} = -1, \quad w_{33}^{12} = 1$$
$$w_{14}^{12} = 1, \quad w_{24}^{12} = -1, \quad w_{34}^{12} = 1$$
$$w_{15}^{12} = -1, \quad w_{25}^{12} = 1, \quad w_{35}^{12} = -1$$
$$w_{16}^{12} = 1, \quad w_{26}^{12} = -1, \quad w_{36}^{12} = -1$$

Inputs for neurons at layer 2 are :

$$v_1^1 = 1, \quad v_1^1 = 1, \quad v_3^1 = -1$$

The weighted sum of these inputs are:

$$u_1^2 = \sum_j^M w_{j1}^{12} v_j^1 = 1 \,.\, 1 + \; 1 \,.\, 1 + \; 1 \,.\, -1 = 1$$

$$u_2^2 = \sum_j^M w_{j2}^{12} v_j^1 = 1 \,.\, 1 + \; 1 \,.\, 1 + -1 \,.\, -1 = 3$$

$$u_3^2 = \sum_j^M w_{j3}^{12} v_j^1 = -1 \,.\, 1 + -1 \,.\, 1 + \; 1 \,.\, -1 = -3$$

$$u_4^2 = \sum_j^M w_{j4}^{12} v_j^1 = 1 \,.\, 1 + -1 \,.\, 1 + \; 1 \,.\, -1 = -1$$

$$u_5^2 = \sum_j^M w_{j5}^{12} v_j^1 = -1 \,.\, 1 + \; 1 \,.\, 1 + -1 \,.\, -1 = \; 1$$

$$u_6^2 = \sum_j^M w_{j6}^{12} v_j^1 = 1 \,.\, 1 + -1 \,.\, 1 + -1 \,.\, -1 = \; 1$$

Since threshold $\theta = M - 1/2 = 3 - 1/2 = 2.5$, the outputs of neurons in the layer 2 are :

$$v_1^2 = g^2(u_1^2) = 0$$

$$v_2^2 = g^2(u_2^2) = 1$$

$$v_3^2 = g^2(u_3^2) = 0$$

$$v_4^2 = g^2(u_4^2) = 0$$

$$v_5^2 = g^2(u_5^2) = 0$$

$$v_6^2 = g^2(u_6^2) = 0$$

After the first phase, we have detected that the second position of the given word is in error.

3.4.2 Error Correction

In the second phase, we use the XOR network with the inputs of the corresponding bit positions of the output of phase 1 (0 1 0 0 0 0) and the received word (1 1 0 1 1 0). Then the XOR network produces the correct codeword (1 0 0 1 1 0).

We have shown the algorithm of a network construction for error detection and correction in linear codes. So far, we can detect and correct a single error in linear codes. In the next chapter, we will develop algorithms to construct neural networks for detection more errors in systematic unidirectional error-detecting codes.

Chapter 4

Systematic Unidirectional Error-Detecting Codes

4.1 Introduction

In this chapter, we construct neural networks for systematic unidirectional double and triple error-detecting codes. Network constructions are developed to find the check symbols which can be used for both encoding and decoding. A heuristic algorithm of neural network construction for systematic unidirectional multiple error-detecting codes is also discussed.

This chapter is organized as follows. In Section 4.2, the fundamental definitions and theorems related to systematic unidirectional codes are given. The constructions of neural networks for systematic unidirectional double and triple error-detecting codes are described in Section 4.3. In Section 4.4 encoding and decoding of the codes are discussed. Examples of network construction of these codes are illustrated in Section 4.5. In Section 4.6, we present a heuristic algorithm of network construction for systematic unidirectional multiple error-detecting codes.

4.2 Preliminaries

In this section, we introduce the background information of error detection in systematic unidirectional codes.

4.2.1 Systematic Unidirectional Codes

We will describe some definitions and theorems from [2][5][6][7][8][24][25][30] as background information concerning our proposed model for detecting errors in systematic unidirectional codes within the framework of neural network.

In systematic codes, the information bits are separately identified from the check bits. Encoding/decoding and data manipulation in systematic codes can be processed in parallel.

Definition 1

A set of errors is said to be *asymmetric* if only one type of errors $1 \rightarrow 0$ (1-error) or $0 \rightarrow 1$ (0-error) can occur, but not both. This error type is assumed to be known a priori.

Definition 2

A set of error is said to be *unidirectional* if both 1-errors and 0-errors can occur in the received words, but in any particular received word, all errors shall be of one type. That is, if multiple errors occur in a given word, then all these errors are 1-errors or 0-errors.

Definition 3

A set of errors is said to be *random* if no specific relationship among the errors exists, such as their being unidirectional, or restricted to contiguous bits.

Definition 4

If both 1-errors and 0-errors appear in a received word with equal probability then the resulting errors are called *symmetric* errors.

From the above definitions, we can see that the asymmetric error class is a subclass of the unidirectional error class, which in turn is a subclass of the symmetric error class. Thus any code capable of correcting/detecting t-symmetric errors is also capable of correcting/ detecting t-unidirectional or t-asymmetric errors, and any code capable of correcting/detecting t-unidirectional errors is also capable of correcting/detecting t-asymmetric errors. However, the converse may not be true[8]. Then we will use the following notations in this paper.

k - number of information bits

r - number of check bits

n = k + r - length of the codeword

k_0 - number of 0's in the information part

k_1 - number of 1's in the information part

Definition 5

$N(X, Y)$ is the number of $1 \rightarrow 0$ crossovers from X to Y.

For example, when $X = (110011)$ and $Y = (001001)$, then $N(X, Y) = 3$ and $N(Y, X) = 1$

Definition 6

$d(X, Y)$, the Hamming distance between X and Y, is given by $d(X, Y) = N(X, Y) + N(Y, X)$.

Definition 7

A word $X = (x_1, x_2, \ldots, x_n)$ is said to *cover* another word $Y = (y_1, y_2, \ldots, y_n)$ whenever $y_i = 1$, $x_i = 1$ for all i = 1,2,...n. We represent X *covers* Y by $X \geq Y$.
For example, when $X = (110011)$ and $Y = (100001)$ then X *covers* Y (i.e., $X \geq Y$)

Definition 8

$X_1, X_2, \ldots, X_n$ is called a *maximal cover* of length n whenever $X_i \leq X_{i+1}$ and there exists no Y which is distinct from X_i and X_{i+1} such that $X_i \leq Y \leq X_{i+1}$ for $i = 1, 2, \ldots, n-1$.
For example, the set {000000,000001,000011,000111,001111,011111,111111} is a maximal cover of length 7.

Definition 9

When neither X *nor* Y covers the other, they are called *unordered.* i.e., if neither $X \geq Y$ nor $Y \geq X$ then X and Y are unordered.
For example, when $X = (110011)$ and $Z = (011001)$ then X and Z are unordered.

Many researchers have developed error correcting codes in many different ways. Basically they are problems in combination of detecting and correcting on symmetric errors, random errors, and unidirectional errors. In this chapter, we use the codes constructed by B. Bose and D. J. Lin [8], which are capable of detecting systematic unidirectional errors. In order to satisfy the necessary and sufficient conditions of being capable to detect systematic unidirectional errors, the following theorems from [8] are described here.

Theorem 1

A code C is capable of detecting t or fewer errors iff the minimum Hamming distance of the codes is at least t+1.

Proof: The proof of this theorem is given in reference [8].

Theorem 2

A code C is capable of detecting t-asymmetric errors iff the following condition is true. For all X, Y in C, either X and Y are unordered or $d(X, Y) \geq t + 1$ when one covers the other, where $d(X, Y)$ is the Hamming distance between X and Y. Further, a code capable of detecting t-asymmetric errors is also capable of detecting t-unidirectional errors.

Proof: The proof of this theorem is given in reference [8].

Theorem 3

A code C is capable of detecting t-unidirectional errors iff the following condition is valid.

For all X, Y in C, either X and Y are unordered or $d(X, Y) \geq t + 1$ when one covers the other.

Proof: The proof of this theorem is given in reference [8].

Based on these theorems, we can construct error detecting codes with neural networks.

In the following sections, we construct our proposed network models of detecting double and triple errors in systematic unidirectional codes assuming that all the necessary conditions prescribed in the above definitions and theorems are satisfied.

4.3 Double and Triple Error Detecting Codes

The double error detecting codes mentioned in this section are systematic codes and error type is unidirectional. Before we construct the neural network, the background information of code construction which is adopted from [8] is discussed.

4.3.1 Code Construction

We consider the unidirectional error detecting codes in the form of systematic codes; i.e., the information bits are separately identified from the check bit. In [8], double, triple and six error-detecting codes with 2, 3,and 4 check bits respectively are described. Then all these codes are also shown to be optimal. Further, the number of check bits r $\geq$ 5 can be used to detect up to $5.2^{r-4}+r-4$ errors. As we mentioned above, double error-detecting codes requires 2 check bits independent of the number of information bits. The check symbol CS for each codeword is generated as follows.

Let k_0 and k_1 be the number of 0's in the information symbol. Then CS = k_0 mod 4. In general, we can find CS = k_0 mod 2^r, where r is the number of check bits. But in particular cases, check symbols are calculated differently according to their needs concerning with less check bits. We can generate the check symbol CS in another way. First we count the number of 1's in the information symbol, then take modulo 2^r, and complement the bits; i.e. CS = (4 - (k_1 mod 4)) mod 4 in the case of double error-detecting codes and CS = (8 - (k_1 mod 8)) mod 8 in the case of

triple error-detecting codes. In general, we can write CS = (2^r - (k_1 mod 2^r)) mod 2^r. There may be different formulas for finding check symbol CS. For r =4 and r ≥ 5, there are different formulations to calculate check symbol CS[8]. For these constructed codes, the capability of detecting t-unidirectional errors is proved in [8].

Before we construct neural networks for double and triple error-detecting codes, we will show examples of generating check bits for those codes. Assume that the number of information bits is 8. Table 4.1 gives the check symbols for the set of information symbols, which is a maximal cover of length 9.

Table 4.1 Example Codewords

Information bits	Check bits
0000 0000	00
0000 0001	11
0000 0011	10
0000 0111	01
0000 1111	00
0001 1111	11
0011 1111	10
0111 1111	01
1111 1111	00

4.3.2 Network for Double Error-Detecting Codes

Here we describe a neural network construction for double error detecting codes. In the following network, there are k neurons in layer 0 where k represents the number of information bits. In layer 1, there are two neurons to represent the number

of check bits. Every neuron in layer 0 is connected with each neuron of layer 1. Neurons in every layer are numbered by a positive integer in consecutive increasing order starting from 1.

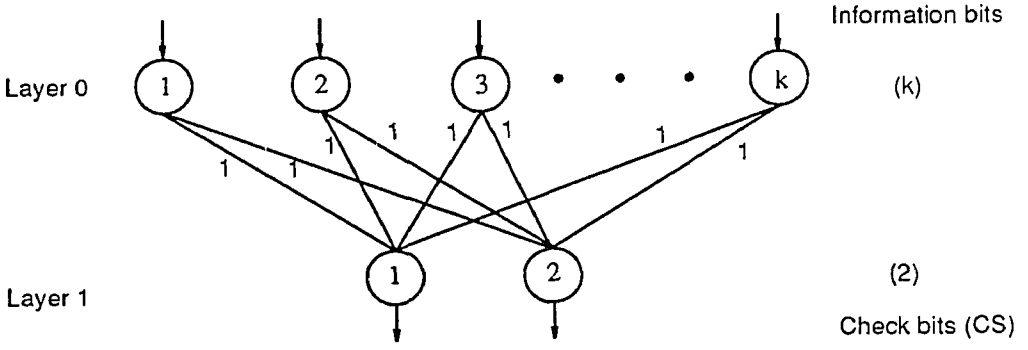

Figure 4.1 Network for Double Error-Detecting Codes

The weight of connection between neurons at layer 0 and layer 1 is simply 1, because we simply wish to count the number of 1's in the information part.

$$w_{ij}^{01} = 1\ ,\ 1 \le i \le \mathrm{k} \text{ and } 1 \le j \le 2.$$

For neuron 1 and neuron 2 of layer 1, the activation functions g^1 and g^2 are defined respectively as follows.

$$g^1(u_i) = ((\ k - u_i) \bmod 2^r) \text{ quo } 2 \qquad (4.1)$$

$$g^2(u_i) = ((\ k - u_i) \bmod 2^r) \bmod 2 \qquad (4.2)$$

where $u_i = \sum_{j=1}^{k} w_{ji}^{01} b_j$, b_j is information bit, r is the number of check bits, and mod and quo are the modulo function and the quotient function respectively. The idea behind the activation functions g^1 and g^2 is to find the check symbol CS. The output value of g^1 and g^2 can only be 1 or 0. Then the required codeword can be formed by

concatenating information part and the check symbol, which is the output of the above network.

4.3.3 Network for Triple Error-Detecting Codes

In this section, we describe a neural network construction for triple error detecting codes.

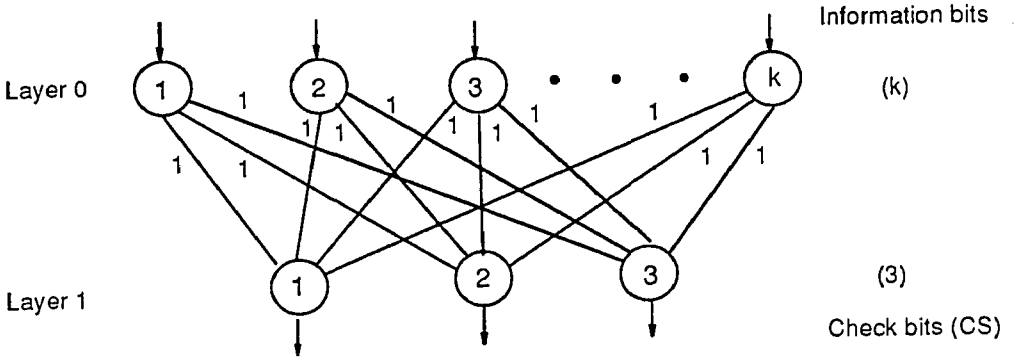

Figure 4.2 Network for Triple Error-Detecting Codes

The network construction for triple error detecting is almost the same as that for double error detecting mentioned in the section 4.3.2. A slight modification is made to the network for double error detecting such as number of neurons and activation functions. In this network, there are k neurons in layer 0 where k represents the number of information bits. In layer 1, there are three neurons to represent the number of check bits. Every neuron in layer 0 is connected with each neuron of layer 1. Neurons in every layer are numbered by a positive integer in consecutive increasing order starting from 1.

The weight of connection between neurons at layer 0 and layer 1 is simply 1. That is to count the number of 1's in the information part.

$$w_{ij}^{01} = 1 \text{ , } 1 \leq i \leq \text{k and } 1 \leq j \leq 3.$$

For neurons 1, 2, and 3 of layer 1, the activation functions g^1 g^2 and g^3 are defined respectively as follows.

$$g^1(u_i) = ((k - u_i) \bmod 2^r) \text{ quo } 2^{r-1}. \quad (4.3)$$

$$g^2(u_i) = ((k - u_i) \bmod 2^{r-1}) \text{ quo } 2^{r-2}. \quad (4.4)$$

$$g^3(u_i) = ((k - u_i) \bmod 2^{r-1}) \bmod 2^{r-2}. \quad (4.5)$$

where $u_i = \sum_{j=1}^{k} w_{ji}^{01} b_j$, b_j is the *jth* information bit, r is the number of check bits and mod and quo are modulo function and quotient function respectively. The activation functions g^1, g^2 and g^3 are used to find the check symbol CS. The output value of g^1, g^2 and g^3 can only be 1 or 0. The required codeword can also be formed by concatenating information part and the output of the above network, like forming a codeword in double error detecting code.

4.4 Encoding and Decoding

In the above double error detecting codes, the check symbol is calculated by k_0 mod 2^r or (2^r - (k_1 mod 2^r)) mod 2^r, where r = 2 and 3. Thus, at the encoder side we need to generate the check symbol, and a circuit is required to calculate the number of 0's mod 2^r. Similarly, to verify whether the received word is error free, we need to find the value of the number 0's mod 2^r in the received word and compare this with the received check symbol. If they match, then the received word is correct, otherwise there must be some error.

For the verification point of view, we need to use our network again to produce new check symbol for the received word. In the matching case, we apply our XOR

network shown in Figure 3.3 of chapter 3. The number of XOR networks is equal to the number of the check symbol. Inputs for XOR networks are the corresponding bits of the new check symbol and the received check symbol. If at least one of XOR network produces the value 1, we can say that there are some errors in the received word.

The following diagram shows the structure of the matching case where $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_n)$ represent the new check symbol of the received word and the received check symbol respectively.

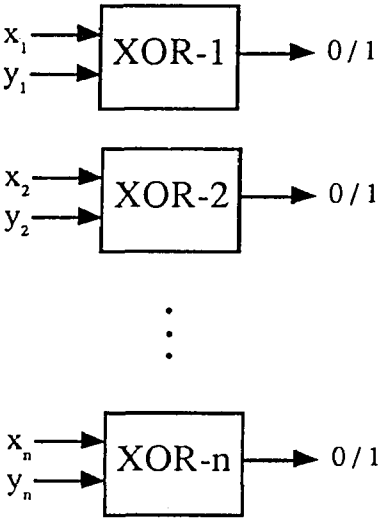

Figure 4.3 The Structure of Matching Case

Our proposed network can be used for either encoding or decoding. In [8], a circuit which generate k_0 mod 4 for the double error detecting codes with k = 8 is described, shown in Figure 4.4. This circuit is implemented using a tree-type r bit 2's complement address. Since our network is constructed with two layers, not only is the network much faster than Bose and Lin's tree-type circuit, but this network can also be used in real time applications. We believe that the proposed network is able

to be modified for multiple error-detecting codes by extending the number of neurons and finding appropriate activation functions. We still need to investigate all unidirectional error detecting and correcting codes.

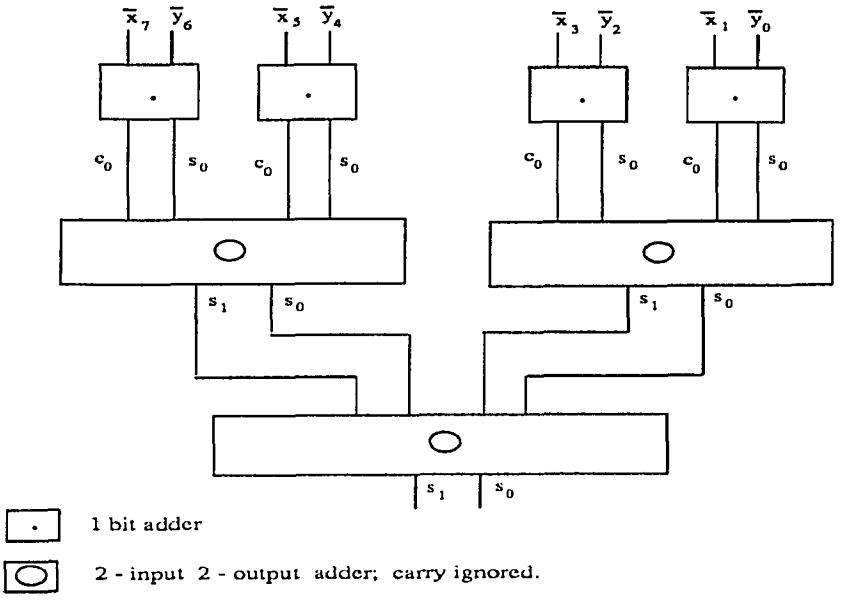

Figure 4.4 Bose and Lin's Check Symbol Generator Circuit for double error-detecting code with k=8

4.5 Illustrative Examples

In this section, we will show how the proposed network works for double and triple error-detecting codes.

4.5.1 Example for Double Error-Detecting Codes

Let the number of information bits be 8. Then we will find the check symbol by using the proposed network. Assume that the information word is 0011 1111.

The weights of the synapse connecting between input layer 0 and layer 1 are :

$$w_{ij}^{01} = 1 \ , \ 1 \leq i \leq 8 \text{ and } 1 \leq j \leq 2.$$

Inputs for layer 1, i.e. bits of the information part, are:

$$b_1 = 1, b_2 = 1, b_3 = 1, b_4 = 1, b_5 = 1, b_6 = 1, b_7 = 0, b_8 = 0$$

According to the proposed network, we need to find the weighted sum of these inputs as follow:

$$u_1 = \sum_{j=1}^{8} w_{j1}^{01} b_j = 1.0 + 1.0 + 1.1 + 1.1 + 1.1 + 1.1 + 1.1 + 1.1 = 6$$

Since all the weights of the synapse between layer 0 and layer 1 have the same value 1, other weighted sum u_2 is also equal to 6.

By applying the functions (4.1) and (4.2) respectively, the outputs of neurons in the layer 1 are :

$$v_1 = g^1(u_1) = 1$$

$$v_2 = g^1(u_2) = 0$$

Thus we have check symbol '10' for the information part 0011 1111. After concatenating these two parts, we will have the codeword 0011 1111 10. At the encoder side, if we receive the word 0011 0101 10 instead of the actual codeword 0011 1111 10, then the new check symbol will be generated as the received check symbol. If we apply the XOR networks for detection, the input will be $X = (x_1 = 0, x_2 = 0)$ and $Y = (y_1 = 1, y_2 = 0)$. Thus one of XOR networks outputs the value '1' and it points out that the received word has some errors.

4.5.2 Example for Triple Error-Detecting Codes

In this section, we will demonstrate the construction of triple error-detecting code by using our proposed network in a similar fashion shown in the above section. Let the number of information bits be 16. Then we will find the check symbol. Here let us assume that the information word is 0000 1111 1111 1111.

The weights of the synapse connecting between input layer 0 and layer 1 are :

$$w_{ij}^{01} = 1 \ , \ 1 \leq i \leq 16 \text{and } 1 \leq j \leq 2.$$

Inputs for layer 1, i.e. bits of the information part, are:

$$b_1 = 1, b_2 = 1, b_3 = 1, b_4 = 1, b_5 = 1, b_6 = 1, b_7 = 1, b_8 = 1,$$
$$b_9 = 1, b_{10} = 1, b_{11} = 1, b_{12} = 1, b_{13} = 0, b_{14} = 0, b_{15} = 0, b_{16} = 0,$$

According to the proposed network, we need to find the weighted sum of these inputs as follows:

$$u_1 = \sum_{j=1}^{16} w_{j1}^{01} b_j = 1.0+1.0+1.0+1.0+1.1+1.1+1.1+1.1+$$
$$1.1+1.1+1.1+1.1+1.1+1.1+1.1+1.1 = 12$$

Since all the weights between layer 0 and layer 1 have the same value 1, other weighted sums u_2 and u_3 are also equal to 12.

By applying functions (4.3), (4.4), and (4.5) respectively, the outputs of neurons in the layer 1 are :

$$v_1 = g^1(u_1) = 1$$

$$v_2 = g^2(u_2) = 0$$

$$v_3 = g^3(u_3) = 0$$

Thus we have check symbol '100' for the information part 0000 1111 1111 1111. After concatenating these two parts, we will have the required codeword 0000 1111 1111 1111 100. At the encoder side, if we receive the word 0011 0111 1111 1111 100 instead of the actual codeword 0000 1111 1111 1111 100 then the new check symbol will be generated as the received check symbol. If we apply the XOR networks for detection, the input will be $X = (x_1 = 0, x_2 = 1, x_3 = 1)$ and $Y = (y_1 = 1, y_2 = 0, y_3 = 0)$. Thus all of the XOR networks outputs the value '1' and it means that we had received the information with errors.

4.6 Multiple Error-Detecting Codes

In the above section, we presented the construction of Neural Networks for systematic unidirectional double and triple error-detecting codes. There may be a case of having more errors in information processing. So many researchers have studied in many different ways to find multiple error correcting and detecting codes. In this section, we will discuss heuristics for the general construction of Neural Network for systematic unidirectional multiple error-detecting codes.

As we mentioned in the previous Section, double and triple error-detecting codes need 2 and 3 check bits respectively. Again according to B. Bose and D. J. Lin[8], 4 check bits are required for 6 unidirectional errors and the systematic codes with 5 or more check bits can detect up to $5 \cdot 2^{r-4} + r - 4$ unidirectional errors where r is the number of check bits. At this point, we can see that the more check bits we

use, the more errors we can detect. In the above Section, neural networks for double error-detecting and triple error-detecting were constructed by using 2 and 3 neurons at layer 1 respectively. So we can say that the number of neurons at layer 1 is equal to the number of check bits in our network construction.

The check symbol CS for each codeword is generated according to the appropriate formula. For instance, CS = (4 - (k_1 mod 4)) mod 4 is used for double error-detecting and CS = (8 - (k_1 mod 8)) mod 8 is used for triple error-detecting, where k_1 is the number of 1's in the information part shown in Section 4.5. There must be appropriate formulas to generate the check symbols for multiple error-detecting codes. According to the value of check symbol, the check bits in binary form are produced by neurons at layer 1 in the network. In the illustrative example 4.3.1, the check symbol 2 ("1 0" in binary) is generated by the formula CS = (4 - (k_1 mod 4)) mod 4 for the information word 0011 1111. These check bits "1 0" are produced by the neurons N_1 and N_2 respectively in the double error-detecting network in Figure 4.1 using the appropriate activation functions. Activation functions have an important role in generating the proper check bits. Finding these functions is the major part of the network construction but it can be done by using the generating function techniques in combinatorics. Therefore we can present an algorithm to construct neural network for multiple error-detecting codes.

Heuristic Algorithm

Begin

Place k neurons at layer 0, where k is the number of information bits

Place r neurons at layer 1, where r is the number of check bits

Connect neurons between layer 0 and layer 1

Give weight "1" for each connection between layer 0 and layer 1

Find the appropriate activation functions for each neuron at layer 1

End

The above algorithm is very general for the construction of network for multiple error-detecting codes. According to the algorithm, only activation functions are required to generate check bits for a particular systematic unidirectional code to construct its specific network.

As discussed earlier, we developed algorithms for double and triple error-detecting codes and a heuristic algorithm of neural network construction for unidirectional multiple error-detecting codes was shown. It is desirable to design algorithms for more general codes. So we will introduce new algorithms of the construction of neural networks for t-Error Correcting/d-Error Detecting (d > t) and All Unidirectional Error Detecting Codes (t-EC/d-ED/AUED) in the next chapter.

Chapter 5

t-Error Correcting/ d-Error Detecting (d > t) and All Unidirectional Error Detecting Codes

5.1 Introduction

We introduced correcting errors in linear codes in Chapter (3) and detecting errors in systematic unidirectional codes in Chapter (4). In Section 4.1, we constructed neural networks for systematic unidirectional double and triple error-detecting codes. In this chapter, we will describe the construction of neural networks for t-Error Correcting(t-EC)/d-Error Detecting(d-ED) (d>t) and All Unidirectional Error Detecting Codes(AUED).

This chapter is organized as follows. In Section 5.2, preliminary information of systematic t-EC/d-ED/AUED codes with d > t are given. Then we also describe the necessary and sufficient conditions for a code to be t-EC/d-ED/AUED with d>t and examples of code construction. In section 5.3, we present our proposed networks for code construction and error detection/correction with the illustrative examples.

5.2 Design of Systematic t-EC/d-ED/AUED Codes with d>t

In this section, we introduce the background information of theory and design of t-error correcting/d-error detecting (d>t) and all unidirectional error detection codes. The design of various forms of t-EC/AUED codes appear in [1][3][4][5][6][9][26][29][30][33][34][36]. "Recently, D.J. Lin and B. Bose[32] have

designed t-EC/d-UED codes with d > t for the case where the number of unidirectional errors, even though large, is limited"[25]. Since t-EC/d-ED/AUED with d > t codes designed in [25] are much more reliable than other codes with respect to redundancy, speed of encoding and decoding, and cost of implementation, we construct neural networks for detecting/correcting those codes. In the following sections, we state the background information related to the design and theory of code construction of Dimitris Nikolos [25].

5.2.1 Necessary And Sufficient Conditions

We will state definitions and theorems from [2][5][6][7][8][[24][25][30] as background information concerning our proposed model for t-EC/d-ED/AUED codes within the framework of neural network. Some definitions and theorems have been mentioned in Section 4.2.2, and we define some more in this section.

We will use the following notations in this chapter.

$L(X)$ - number of 0's in X

$|A|$ - the cardinality of a code A

Definition 1

$\Delta(X, Y) = \max\{N(X, Y), N(Y, X)\}$: represents the asymmetric distance between X and Y.

Definition 2

$\Delta = \min_{\substack{X,Y \in A \\ X \neq Y}} \Delta(X, Y)$: represents the minimum asymmetric distance of a code A.

In this section, we use the codes constructed by Dimitris Nikolos[25] which are capable of t-error correcting/d-error detecting (d>t)and all unidirectional error detecting codes. In order to satisfy the necessary and sufficient conditions for a code to be t-EC/d-ED/AUED with d > t, the following theorems from [25] are described.

The following theorem is mentioned in [5][30] for a t-EC/AUED (t- Error Correcting/ All Unidirectional Error Detecting) code.

Theorem 1

A code C is t-ED/AUED iff it satisfies the following condition:

$\forall\ X, Y \in C$ with $X \neq Y$, $N(X, Y) \geq t + 1$ and $N(Y, X) \geq t + 1$

Proof: The proof of this theorem is given in the references [5] and [30].

A stronger result for a code C is given in the following theorem[5][25][29].

Theorem 2

For all distinct $X, Y \in C$ if $N(X, Y) \geq t + 1$ and $N(Y, X) \geq t + 1$ then C is capable of correcting t or fewer symmetric errors, detecting t+1 symmetric errors and also detecting all (t+2 or more) unidirectional errors.

Proof: The proof is given in the references [5] and [30].

Theorem 3

A code C is t-EC/d-ED/AUED with d>t+1 iff it satisfies the following conditions: For all distinct $X, Y \in C$, $d(X, Y) \geq t + d + 1$, $N(X, Y) \geq t + 1$ and $N(Y, X) \geq t + 1$.

Proof: The proof of this theorem is given in the reference [25].

Based on these theorems, we can construct t-EC/d-ED/AUED codes with neural networks. Before constructing a network, a method for constructing systematic t-EC/d-ED/AUED codes with d > t from [25] will be described in the next section.

5.2.2 Code Construction

Let F be a systematic t-error correcting and d-error detecting parity check code with length n'. Also, let $A_1, A_2, \ldots, A_k$ with $1 \le k \le t+1$ be codes with lengths $r_1, r_2, \ldots, r_k$ and asymmetric distances $\Delta_1, \Delta_2, .., \Delta_k$ such that $\Sigma_{f=1}^{k} \Delta_f = t+1$.

There will be two possible cases at this point for a code A_j; $\Delta_j = 1$ and $\Delta_j > 1$.

Case I :

For any code A_j with $\Delta_j = 1$, we will use the binary representation of the numbers $0, 1, 2, \ldots, 2^{\lfloor \log(n'+1) \rfloor - a_j} - 1$; where the value of a_j is such that $2^{a_j} \le d - t + 1 + 2.\ \Sigma_{f=1}^{j-1} \Delta_f < 2^{a_j+1}$; as a row in the matrix M_j, $|A_j| \times r_j$. That is, row m is the binary representation of m in the matrix M_j.

Then the cardinality of the code will be $|A_j| = 2^{\lfloor \log(n'+1) \rfloor - a_j}$ and its length $r_j = \lfloor log(n'+1) \rfloor - a_j$.

Case II :

For any code A_i with $\Delta_i > 1$, the rows of the matrix M_i, $|A_i| \times r_i$ are the codewords in the order of nondescending weights $W(X)$, where X is a row in the matrix M_i.

The cardinality of A_i, $|A_i| \ge \lfloor (n'+1)/(d - t + 1 + 2.\Sigma_{f=1}^{j-1} \Delta_f) \rfloor$.

The codewords will have the form

$$X\ R_{1,i_1}\ R_{2,i_2} \cdots R_{k,i_k}$$

i.e., each codeword is the concatenation of $X, R_{1,i_1}, R_{2,i_2}, \ldots R_{k,i_k}$. X represents the encoding of the given information bits.

For $\Delta_j \neq 1$, R_{j,i_j} is the row i_j of matrix M_j with $i_j = \lfloor L(X)/(d-t+1+2.\Sigma_{f=1}^{j-1}\Delta_f) \rfloor$, where $L(X)$ denotes the number of 0's in X.

For $\Delta_j = 1$, R_{j,i_j} is the row i_j of the matrix M_j with $i_j = \lfloor L(X)/2^{a_j} \rfloor$, where the value of a_j satisfies the relation $2^{a_j} \leq d-t+2.\Sigma_{f=1}^{j-1}\Delta_f < 2^{a_j+1}$.

For example, let us assume that $t = 2$, and $d = 8$. According to the technique described in the above section, there will be four different 2-EC/8-ED/AUED codes as shown below.

- One code contains codewords of the form

  XR_{1,i_1} where $R_{1,i_1} \in A_1$ and $\Delta_1 = 3$.

- Two codes contain codewords of the form

  $XR_{1,i_1}R_{2,i_2}$ where $R_{1,i_1} \in A_1$, $R_{2,i_2} \in A_2$ and $\Delta_1 = 1$, $\Delta_2 = 2$ or $\Delta_1 = 2$ and $\Delta_2 = 1$.

- One code contains codewords of the form

  $XR_{1,i_1}R_{2,i_2}R_{3,i_3}$ where $R_{j,i_j} \in A_j$, and $\Delta_j = 1$ for j=1,2,3.

Similarly, for $t = 3$ and $d = 8$, we will have eight different 3-EC/8-ED/AUED codes.

• One code contains codewords of the form

XR_{1,i_1} where $R_{1,i_1} \in A_1$ and $\Delta_1 = 4$.

• Three codes contain codewords of the form

$XR_{1,i_1}R_{2,i_2}$ where $R_{1,i_1} \in A_1$, $R_{2,i_2} \in A_2$ and

$\Delta_1 = 1, \Delta_2 = 3$ or $\Delta_1 = 3, \Delta_2 = 1$ or $\Delta_1 = \Delta_2 = 2$.

• Three codes contain codewords of the form

$XR_{1,i_1}R_{2,i_2}R_{3,i_3}$ where $R_{1,i_1} \in A_1$, $R_{2,i_2} \in A_2$, $R_{3,i_3} \in A_3$, and

$\Delta_1 = \Delta_2 = 1, \Delta_3 = 2$ or $\Delta_1 = 1, \Delta_2 = 2, \Delta_3 = 1$ or $\Delta_1 = 2, \Delta_2 = \Delta_3 = 1$.

• One code contains codewords of the form

$XR_{1,i_1}R_{2,i_2}R_{3,i_3}R_{4,i_4}$ where $R_{j,i_j} \in A_j$, and $\Delta_j = 1$ for $j = 1,2,3,4$.

In the above forms of codewords, some are concatenation of information bits and one group of check bits and some are concatenation of information bits and more than one group of check bits. The main idea for all the codewords is that they are the forms of concatenation of information part and check symbol part.

5.2.3 Examples of Codes

In this section, we will show examples of code construction of t-EC/d-ED/AUED, which are adopted from [25]. First of all, we must have a code F to use as a systematic parity check code. At this point, we assume that F has codewords of length 16 bits and for all distinct $X, Y \in F$, $D(X, Y) \geq 8$. According to the method shown above we can construct four different 2-EC/5-ED/AUED codes. In the following example codes, we use Table 5.1 adopted from [25][35] for the purpose of determining the length of codes, A_j.

Table 5.1
Bounds of the Cardinality of Asymmetric Error Correcting Codes

n	Δ=2	Δ=3	Δ=4	Δ=5
5	6	2	2	2
6	12	4	2	2
7	18	4	2	2
8	36	7	4	2
9	62	12	4	2
10	108-117	18	6	4
11	174-210	30-32	8	4
12	316-410	54-63	12	4
13	586-786	98-114	18	6
14	1096-1500	186-218	30-34	8
15	2048-2828	266-398	44-50	12
16	3856-5430	364-739	66-90	16
17	7296-10374	647-1279	122-168	26
18	13798-19898	1218-2380	234-320	36-44
19	26216-38008	2050-4242	450-616	46-76
20	49940-73174	2564-8069	860-1144	54-134
21	95326-140798	4251-14374	1628-2134	62-229
22	182362-271953	8450-26679	3072-4116	88-423
23	349536-523586	16388-50200	4096-7346	133-745

5.2.3.1 Algorithm for Code I

According to the technique shown in Section 5.2.2, one code contains codewords of the form XR_{1,i_1} , where $R_{1,i_1} \in A_1$ and $\Delta_1 = 3$. Then the cardinality of A_1 is 5 since $|A_1| \geq \lceil (n'+1)/(d-t+1) \rceil$,where $n' = 16, d = 5$, and $t = 2$. According to Table 5.1 from [35] a code with five codewords and $\Delta \geq 3$ will have length greater than or equal to 8. Here we use the code with codewords (00000000), (00000111), (00111000), (00111111) and (11110100). So we will have the matrix M_1 as follows:

$$M_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Using the technique described in Section 5.2.2, we will have all the codewords of the 2-EC/5-ED/AUED code as shown in Table 5.2.

Table 5.2 2-EC/5-ED/AUED Code I

X	R_{1,i_1}
0 0 1 1 1 1 1 1 0 0 0 0 0 0 1	0 0 1 1 1 0 0 0
0 1 0 0 0 0 1 1 1 1 1 1 0 0 0 1	0 0 1 1 1 0 0 0
1 0 0 0 0 0 0 0 1 1 1 1 1 1 1	0 0 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 1 1 1 1 1 1
0 1 1 1 1 0 0 0 1 1 1 0 0 0 0	0 0 1 1 1 0 0 0
1 0 1 1 1 1 1 1 1 1 1 1 1 1 0	0 0 0 0 0 0 0 0
1 1 0 0 0 0 1 1 1 0 0 0 1 1 1 0	0 0 1 1 1 0 0 0
1 1 1 1 1 0 0 0 0 0 0 1 1 1 1	0 0 0 0 0 1 1 1

5.2.3.2 Algorithm for Code II

As we have shown with an example in Section 5.2.3.1, we can construct another 2-EC/5-ED/AUED code with the codewords of the form $XR_{1,i_1}R_{2,i_2}$, where $R_{1,i_1} \in A_1$ and $R_{2,i_2} \in A_2$, with $\Delta_1 = 1$ and $\Delta_2 = 2$. Since $\Delta_1 = 1$, the cardinality of A_1, $|A_1| = 2^{\lfloor log(n'+1) \rfloor - a_1}$, where the value of a_1 is such that $2^{a_1} \leq d - t + 1 < 2^{a_1+1}$.

Thus, $a_1 = 2$ and $|A_1| = 8$. According to the method described in Section 5.2.2, the matrix M_1 will be

$$M_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

We still need to compute the cardinality of A_2. According to the method shown in Section 5.2.2, $|A_2| \geq |(n'+1)/(d-t+1+2)| = 3$. So there will be three codewords and length of each codeword is equal to 4. Then we will get the matrix M_2 as follows:

$$M_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Now we present all the codewords of a 2-EC/5-ED/AUED code with the form $XR_{1,i_1}R_{2,i_2}$, where $R_{1,i_1} \in A_1$ and $R_{2,i_2} \in A_2$, with $\Delta_1 = 1$ and $\Delta_2 = 2$ in Table 5.3.

Table 5.3 2-EC/5-ED/AUED Code II

X	R_{1,i_1}	R_{2,i_2}
001 111111000000 1	010	0011
010 000111111000 1	010	0011
100 000000111111 1	010	0011
000 000000000000 0	100	1111
011 111000111000 0	010	0011
101 111111111111 0	000	0000
110 000111000111 0	010	0011
111 111000000111 1	001	0011

5.2.3.3 Algorithm for Code III

According to the method mentioned in Section 5.2.2, there is a similar code like the one in Section 5.2.3.2. This code contains codewords with the form $XR_{1,i_1}R_{2,i_2}$, where $R_{1,i_1} \in A_1$ and $R_{2,i_2} \in A_2$, with $\Delta_1 = 2$ and $\Delta_2 = 1$.

As we have seen the method in Section 5.2.2, the cardinality of A_1, $|A_1| \geq |(n'+1)/(d-t+1+2)| = 5$. We will take the code with codewords

(0 0 0 0 0), (0 0 0 1 1), (0 1 1 1 0), (1 1 0 0 0), and (1 1 1 1 1) as A_1. Thus we get the matrix M_1 as follows:

$$M_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Since $\Delta_2 = 1$, the cardinality of A_2 , $|A_2| = 2^{\lceil log(n'+1) \rceil - a_2}$, where the value of a_2 is such that $2^{a_2} \leq d - t + 1 + 2.2 < 2^{a_2+1}$.

Thus, $a_2 = 3$ and $|A_2| = 4$. According to the method described in Section 5.2.2, the matrix M_2 will be

$$M_2 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

In Table 5.4, all the codewords of a 2-EC/5-ED/AUED code with the form of $XR_{1,i_1}R_{2,i_2}$, where $R_{1,i_1} \in A_1$ and $R_{2,i_2} \in A_2$, with $\Delta_1 = 2$ and $\Delta_2 = 1$ are presented.

Table 5.4 2-EC/5-ED/AUED Code III

X	R_{1,i_1}	R_{2,i_2}
0 0 1 1 1 1 1 1 0 0 0 0 0 0 1	1 1 0 0 0	0 1
0 1 0 0 0 0 1 1 1 1 1 0 0 0 1	1 1 0 0 0	0 1
1 0 0 0 0 0 0 0 1 1 1 1 1 1 1	1 1 0 0 0	0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1 1 1 1 1	1 0
0 1 1 1 1 0 0 0 1 1 1 0 0 0 0	1 1 0 0 0	0 1
1 0 1 1 1 1 1 1 1 1 1 1 1 1 0	0 0 0 0 0	0 0
1 1 0 0 0 0 1 1 1 0 0 0 1 1 0	1 1 0 0 0	0 1
1 1 1 1 1 0 0 0 0 0 0 1 1 1 1	0 0 0 1 1	0 0

5.2.3.4 Algorithm for Code IV

Here we will show the construction of a 2-EC/5-ED/AUED code with the form $XR_{1,i_1}R_{2,i_2}R_{3,i_3}$, where $R_{j,i_j} \in A_j$ and $\Delta_j = 1$ for j=1,2,3.

Since $\Delta_j = 1$, the cardinality of A_j , $|A_j| = 2^{\lceil log(n'+1) \rceil - a_j}$, where the value of a_j is such that $2^{a_j} \le d - t + 1 + 2.2 < 2^{a_j+1}$. Thus, $a_1 = a_2 = 2, a_3 = 3$ and $|A_1| = 8, |A_2| = 8$ and $|A_3| = 4$. Then

$$M_1 = M_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$M_3 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

In Table 5.5, all the codewords of a 2-EC/5-ED/AUED code with the form of $XR_{1,i_1}R_{2,i_2}R_{3,i_3}$, where $R_{j,i_j} \in A_j$ and $\Delta_j = 1$ for j=1,2,3 are presented.

Table 5.5 2-EC/5-ED/AUED Code IV

X	R_{1,i_1}	R_{2,i_2}	R_{3,i_3}
001 111111000000 1	010	010	01
010 000111111000 1	010	010	01
100 000000111111 1	010	010	01
000 000000000000 0	100	100	10
011 111000111000 0	010	010	01
101 111111111111 0	000	000	00
110 000111000111 0	010	010	01
111 111000000111 1	001	001	00

5.2.4 A Scheme for Error Detection/Correction

In this section, we will present an error detection/correction algorithm for the t-EC/d-ED/AUED codes, which is given by Dimitris Nikolos[25]. We will describe the network construction of this algorithm in a later section.

Let $Q = XR_{1,i_1}R_{2,i_2}\cdots R_{k,i_k}$ be an error-free codeword of an t-EC/d-ED/AUED code and $Q' = X'R'_{1,i_1}R'_{2,i_2}\cdots R'_{k,i_k}$ be the received codeword which has some errors.

The formal algorithm[25] for error detecting and correcting t-EC/d-ED/AUED codes is as follows:

begin

Let H be the parity check matrix corresponding to the systematic parity check code F, where F has been defined in Section 5.2.

Compute syndrome S of X', that is, $S = H.X'^T$.

Let S correspond to g multiplicity error.

if $g > t$ **then** the error is only detectable and stop

else correct X' using the correcting procedure in the parity check code obtaining X'' as the resulting word.

Compute R''_{j,i_j} for $j = 1, 2, \ldots k$ corresponding to X''.

Let $Q'' = X''R''_{1,i_1}R''_{2,i_2}\cdots R''_{k,i_k}$.

if $d(Q', Q'') \leq t$ **then** Q'' is a correct codeword

else errors are only detectable .i.e. errors > t occurred.

end

Figure 5.1 Formal Algorithm of Error Detection/Correction

5.2.4.1 Example

Here we will show how error detection/correction works according to the above algorithm. In this example[25], 2-EC/5-ED/AUED Code I shown in Section 5.2.3.1 is assumed to be the given code. Let the parity check matrix be

$$H = \begin{bmatrix}
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}.$$

Suppose a correct codeword

$Q = 0\,0\,1 \quad 1\,1\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,1 \quad 0\,0\,1\,1\,1\,0\,0\,0.$

$\qquad\qquad X \qquad\qquad\qquad\qquad R_{1,2}$

Then we assume that the received word would be

$Q' = 1\,0\,1 \quad 1\,1\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,1 \quad 0\,0\,0\,1\,1\,0\,0\,0.$

$\qquad\qquad X' \qquad\qquad\qquad\qquad R'_{1,2}$

According to the algorithm,

the syndrome $S = H.X'^{T} = [0000000011111111]^{T}$.

Since the value of syndrome S is equal to the first column of the parity check matrix H, a single error has occurred in the first position of X' in the received word Q'. Then after correcting the error, the resulting word

$X'' = (0011111110000001)$

Then we will have the new check bits R''_{1,i_1} corresponding to X'' using the formula $i_1 = \lfloor L(X)/(d-t+1) \rfloor = \lfloor 8/(5-2+1) \rfloor = 2$. Thus $R''_{1,i_1} = R_{1,2} = (0\ 0\ 1\ 1\ 1\ 0\ 0\ 0)$.

So $Q'' = 0\ 0\ 1\ \ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ \ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0$

In order to check whether the resulting word is the correct word or not, we need to compute the Hamming distance between Q' and Q''.

$$d(Q', Q'') = 2 \leq t = 2.$$

So we can say that Q'' is the correct word.

5.3 The Neural Network

In this section, we present neural networks for code construction and error detection/correction. Having described code construction and error detection/correction in the above section, we will proceed to construct networks according to the described methods. Though the code constructing method in Section 5.2 is complicated, our proposed network construction is simple and easy to understand. Mostly the network constructions in our error correction paradigm are similar to each other.

5.3.1 Code Construction and Compilation

As we have seen the method of the t-EC/d-ED/AUED code construction in Section 5.2.2, codewords are formed by concatenation of information bits and one or more groups of check bits depending on how we want to construct code. For example, we need to decide what the length of a codeword should be. From our point of view of network construction, every code is treated in the same way except the

number of networks and its applications. So we will discuss a general algorithm for network construction and demonstrate some particular cases.

5.3.1.1 A New Algorithm

According to the method shown in Section 5.2.2, we know that the codeword is the form of $XR_{1,i_1}R_{2,i_2}\cdots R_{k,i_k}$ where X represents the information bits and R_{j,i_j}, $1 \le j \le k$, is the check bits. As we presented the method of t-EC/d-Ed/AUED code construction in Section 5.2, we need to find R_{j,i_j} which is the row i_j of the matrix M_j. We assume that matrix M_j has been already known after computing the cardinality of A_j and Δ values. R_{j,i_j} can be calculated in two different ways depending on asymmetric distance Δ value which may be 1 or greater than 1. In each case, the row i_j has to be computed by using either the formula $i_j = \lfloor L(X) / (d - t + 1 + 2.\Sigma_{f=1}^{j-1}\Delta_f) \rfloor$ or $i_j = \lfloor L(X) / 2^{a_j} \rfloor$, where the value of a_j satisfies the relation $2^{a_j} \le d - t + 2.\Sigma_{f=1}^{j-1}\Delta_f < 2^{a_j+1}$ with respect to the value of Δ.

Here we will present a general algorithm of code construction in the framework of neural computing. Before we describe the algorithm, the following notations will be used in the following algorithm.

k = length of the information bits X

n = number of columns in the matrix M_j

m = number of rows in the matrix M_j

$Z = d - t + 1 + 2.\Sigma_{f=1}^{j-1}\Delta_f$ for $\Delta > 1$ and

$= 2^{a_j}$ for $\Delta = 1$.

The network is shown in Figure 5.2. There are k inputs to the network, which consists of associative memories comprising of two layers of neurons. In the hidden layer (also referred to as layer 1), there are m neurons which represent the number of rows in the matrix M_j which is assumed to be given. In layer 1, neurons are named $\eta_1, \eta_2, \ldots, \eta_m$, going from left to right. Every input is connected with each neuron of layer 1. Layer 2 has n neurons which will produce the value of R_{j,i_j}. Similarly, neurons at layer 2 have names $\zeta_1, \zeta_2, \ldots, \zeta_n$, going from left to right. Every neuron of layer 1 is connected to every neuron of layer 2.

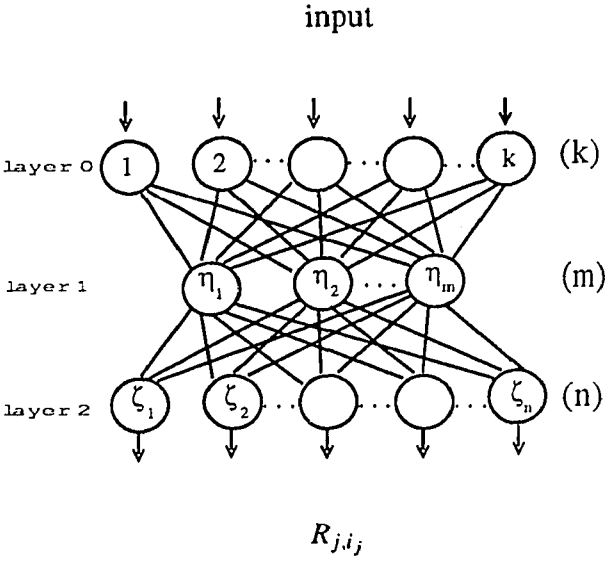

Figure 5.2 Network for Code Construction

We denote the weight w_{ij}^{01} of the connection of the *ith* neuron of the input layer with the *jth* neuron of layer 1.

$$w_{ij}^{01} = 1 \text{ , } 1 \leq i \leq m \text{ and } 1 \leq j \leq n$$

We use w_{ij}^{12} to denote the weight of the connection of the *ith* neuron of layer 1 with the *jth* neuron of layer 2. The values of w_{ij}^{12} are assigned by the elements of the matrix M_j in the following way. In this case, we denote that α_{ij} is the element at the *ith* row and the *jth* column of the matrix M_j.

i.e. For $1 \le i \le m$, $1 \le j \le n$.

$$w_{ij}^{12} = \alpha_{m-i+1,j}$$

For each neuron of layer 1, the hard limiter activation function [Figure 5.3a] is used as the activation function, while the threshold logic function[10] [Figure 5.3b] is used for the neurons of layer 2 [10][22].

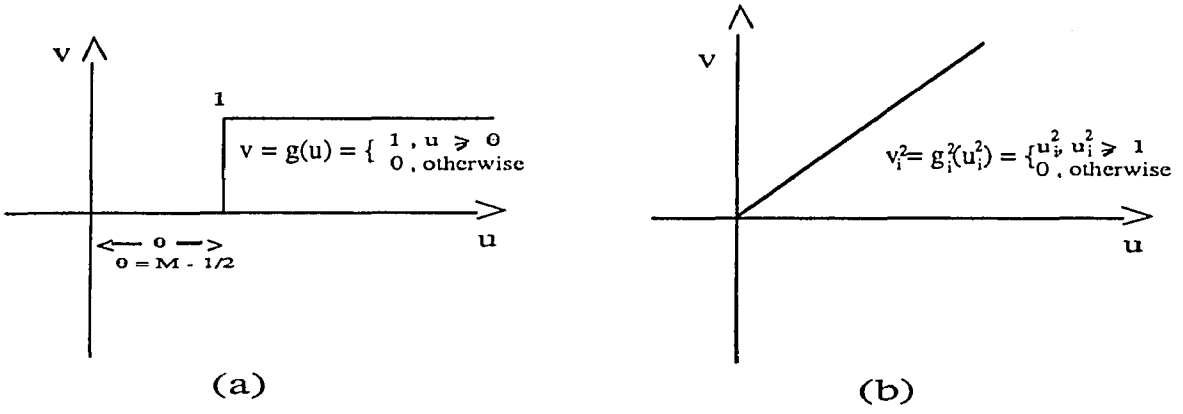

Figure 5.3 Activation Functions Used in the Network

In the network, the information part of the received word is passed through the first layer and we allow the network to progress until it falls into a stable situation. In this case, the output of layer 1 determines the row number of the matrix M_j and layer 2 produces the appropriate check bits for the given information part X.

We introduce the following variables and activation functions to show how our network performs.

(1) The initial input v_j^0, $1 \le j \le N$

(2) The output of neuron t in the layer 1 v_t^1, $1 \le t \le M$

(3) The output of neuron i in the layer 2 v_i^2, $1 \le i \le N$

Let g_t^1 and g_j^2 be the activation functions for neurons of layer 1 and layer 2 respectively, where $1 \le t \le m$ and $1 \le j \le n$. In other words, g_t^1 is the activation function for neuron η_t of layer 1 while g_j^2 is for neuron ζ_j of layer 2. g_t^1 is a hard limiter activation function on the weighted sum of given inputs v_j^0, where $1 \le j \le n$.

Let $u_t^1 = \sum_j^k w_{jt}^{01} v_j^0$, $1 \le t \le m$ and $v_t^1 = g_t^1(u_t^1)$,

$$i.e. \quad v_t^1 = g_t^1(u_t^1) = \begin{cases} 1, & \text{if } S = m - t + 1 \\ 0, & otherwise \end{cases} \qquad (5.1)$$

where $S = \lfloor (k - u_t^1)/Z \rfloor$.

The output values of the neurons of layer 2 are determined by the threshold logic function g_i^2 [10][22].

Let $u_i^2 = \sum_j^m w_{ji}^{12} v_j^1$, $1 \le i \le n$ then we have,

$$i.e. \quad v_i^2 = g_i^2(u_i^2) = \begin{cases} u_i^2, & \text{if } u_i^2 \ge 0 \\ 0, & otherwise \end{cases} \qquad (5.2)$$

In this function, the output of g_i^2 will be either 0 or 1 since the value of u_i^2 is 0 or 1.

5.3.2 Illustrative Examples

Here we will demonstrate our network with illustrative examples. We will use examples of Code I and Code II shown in Section 5.2.3.1 and Section 5.2.3.2 respectively.

5.3.2.1 Example 1

In Section 5.2.3.1, we discussed 2-EC/5-ED/AUED code which contains codewords in the form XR_{1,i_1} where $R_{1,i_1} \in A_1$ and $\Delta_1 = 3$. According to Section 5.2.3.1, we know that the length of the information bits X, $n' = 16$, and $d = 5, t = 2$. Then the matrix M_1 is as follows:

$$M_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

From the above facts, we can illustrate the flow of our network in the following way.

If the information bits $X = 0011111110000001$, are received we can find the appropriate check bits, R_{1,i_1}. In this case, since the number of rows in the matrix M_1 is 5 and the number of columns is 8, we have $m = 5$ and $n = 8$. Also the length of the information bits, k is 16. Since $\Delta_1 = 3 > 1$, then $Z = d - t + 1 = 5 - 2 + 1 = 4$

The weights of the connections between input layer 0 and layer 1

$w_{ij}^{01} = 1$, where $i = 1, 2, \ldots, k$ and $j = 1, 2, \ldots, m$.

Inputs for the layer 0, i.e. bits of the word received, are

$v_1^0 = 0,\ v_2^0 = 0,\ v_3^0 = 1,\ v_4^0 = 1,\ v_5^0 = 1,\ v_6^0 = 1,\ v_7^0 = 1,\ v_8^0 = 1,$

$v_9^0 = 1,\ v_{10}^0 = 0,\ v_{11}^0 = 0,\ v_{12}^0 = 0,\ v_{13}^0 = 0,\ v_{14}^0 = 0,\ v_{15}^0 = 0,\ v_{16}^0 = 1$

According to the proposed network, we need to find the weighted sum of these inputs. Since the weight of the each connection between layer 0 and layer 1 is 1, the weighted sum of these inputs will be the same.

Thus

$$u_t^1 = \sum_j^n w_{jt}^{01} v_j^0 = 1.0 + 1.0 + 1.1 + 1.1 + 1.1 + 1.1 + 1.1 + 1.1 + 1.1 +$$

$$1.0 + 1.0 + 1.0 + 1.0 + 1.0 + 1.0 + 1.1 = 8$$

Since we get $Z = 4$ and $S = \lfloor (k - u_t^1) / Z \rfloor$, $S = \lfloor (16 - 8)/4 \rfloor = 2$, then we can compute the outputs of neurons in the layer 1 using equation (5.1) as follows:

$$v_1^1 = g_1^1(u_1^1) = 0$$

$$v_2^1 = g_2^1(u_2^1) = 0$$

$$v_3^1 = g_3^1(u_3^1) = 1$$

$$v_4^1 = g_4^1(u_4^1) = 0$$

$$v_5^1 = g_5^1(u_5^1) = 0$$

The weights of synapse connecting layer 1 and layer 2 are :

$$w_{11}^{12} = 0, w_{21}^{12} = 1, w_{31}^{12} = 0, w_{41}^{12} = 0, w_{51}^{12} = 0$$

$$w_{12}^{12} = 0, w_{22}^{12} = 1, w_{32}^{12} = 0, w_{42}^{12} = 0, w_{52}^{12} = 0$$

$$w_{13}^{12} = 1, w_{23}^{12} = 1, w_{33}^{12} = 1, w_{43}^{12} = 0, w_{52}^{12} = 0$$

$$w_{14}^{12} = 1, w_{24}^{12} = 1, w_{34}^{12} = 1, w_{44}^{12} = 0, w_{54}^{12} = 0$$

$$w_{15}^{12} = 1, w_{25}^{12} = 0, w_{35}^{12} = 1, w_{45}^{12} = 0, w_{55}^{12} = 0$$

$$w_{16}^{12} = 1, w_{26}^{12} = 1, w_{36}^{12} = 0, w_{46}^{12} = 1, w_{56}^{12} = 0$$

$$w_{17}^{12} = 1, w_{27}^{12} = 0, w_{37}^{12} = 0, w_{47}^{12} = 1, w_{57}^{12} = 0$$

$$w_{18}^{12} = 1, w_{28}^{12} = 0, w_{38}^{12} = 0, w_{48}^{12} = 1, w_{58}^{12} = 0$$

Inputs for neurons at layer 2 are :

$$v_1^1 = 0, v_2^1 = 0, v_3^1 = 1, v_4^1 = 0, v_5^1 = 0$$

The weighted sum of these inputs are:

$$u_1^2 = \sum_j^m w_{j1}^{12} v_j^1 = 0 \,.\, 0 + 1 \,.\, 0 + 0 \,.\, 1 + 0 \,.\, 0 + 0 \,.\, 0 = 0$$

$$u_2^2 = \sum_j^m w_{j2}^{12} v_j^1 = 0 \,.\, 0 + 1 \,.\, 0 + 0 \,.\, 1 + 0 \,.\, 0 + 0 \,.\, 0 = 0$$

$$u_3^2 = \sum_j^m w_{j3}^{12} v_j^1 = 1 \,.\, 0 + 1 \,.\, 0 + 1 \,.\, 1 + 0 \,.\, 0 + 0 \,.\, 0 = 1$$

$$u_4^2 = \sum_j^m w_{j4}^{12} v_j^1 = 1 \,.\, 0 + 1 \,.\, 0 + 1 \,.\, 1 + 0 \,.\, 0 + 0 \,.\, 0 = 1$$

$$u_5^2 = \sum_j^m w_{j5}^{12} v_j^1 = 1 \,.\, 0 + 0 \,.\, 0 + 1 \,.\, 1 + 0 \,.\, 0 + 0 \,.\, 0 = 1$$

$$u_6^2 = \sum_j^m w_{j6}^{12} v_j^1 = 1 \,.\, 0 + 1 \,.\, 0 + 0 \,.\, 1 + 1 \,.\, 0 + 0 \,.\, 0 = 0$$

$$u_7^2 = \sum_j^m w_{j7}^{12} v_j^1 = 1 \,.\, 0 + 0 \,.\, 0 + 0 \,.\, 1 + 1 \,.\, 0 + 0 \,.\, 0 = 0$$

$$u_8^2 = \sum_j^m w_{j8}^{12} v_j^1 = 1 \,.\, 0 + 0 \,.\, 0 + 0 \,.\, 1 + 1 \,.\, 0 + 0 \,.\, 0 = 0$$

Since we have defined the threshold logic function as an activation function (5.2) for each neuron, the outputs of neurons in the layer 2 are :

$$v_1^2 = g_1^2(u_1^2) = 0\ ,\ v_2^2 = g_2^2(u_2^2) = 0\ ,\ v_3^2 = g_3^2(u_3^2) = 1\ ,\ v_4^2 = g_4^2(u_4^2) = 1$$

$$v_5^2 = g_5^2(u_5^2) = 1\ ,\ v_6^2 = g_6^2(u_6^2) = 0\ ,\ v_7^2 = g_7^2(u_7^2) = 0\ ,\ v_8^2 = g_8^2(u_8^2) = 0$$

So the check bits R_{1,i_1} are 0 0 1 1 1 0 0 0 for the given information bits X ; 0011111110000001. Thus the required codewords will be

001 1111110000001 00111000

which can be checked using Table 5.2 2-EC/5-ED/AUED Code I.

In this way, we can construct the required t-EC/d-ED/AUED code which contains codewords of the form XR_{1,i_1} .

5.3.2.2 Example 2

In this example, we present the demonstration of construction of 2-EC/5-ED/AUED code which contains codewords of the form $XR_{1,i_1}R_{2,i_2}$ where $R_{1,i_1} \in A_1$, $R_{2,i_2} \in A_2$ and with $\Delta_1 = 1$ and $\Delta_2 = 2$. A formal way for constructing this code has been illustrated in Section 5.2.3.2 and named as Code II. According to Section 5.2.3.2, we know that the length of the information bits X , $n' = 16$, and $d = 5, t = 2$. Since $\Delta_1 = 1$, we compute the cardinality of A_1 , $|A_1| = 8$ by using the formula $|A| = 2^{\lceil log(n'+1) \rceil - a_1}$, where a_1 is such that $2^{a_1} \le d - t + 1 < 2^{a_1+1}$, i.e , $a_1 = 2$. So the binary representation of the numbers $0, 1, 2, \ldots, 2^{\lceil log(n'+1) \rceil - a_1} - 1$ are the rows in the $|A_1| \times r_1$, matrix M_1. In this case, $r_1 = 3$ which is computed according to Section 5.2.2. Thus the matrix M_1 will be

$$M_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Similarly, according to Section 5.2.2 and Section 5.2.3.2, we have $|A_2| = 3$ by the formula $|A| \geq \lceil (n'+1) / (d - t + 1 + 2.\Sigma_{f=1}^{j-1}\Delta_f) \rceil$. So the $|A_2| \times r_2$ matrix M_2 will be

$$M_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

From the above facts, we will construct a network for finding two groups of check bits R_{1,i_1} and R_{2,i_2} . Let the information bits X of the received word be 100 0000001111111. we initially find the appropriate check bits, R_{1,i_1} for X. In this case, since the number of rows in the matrix M_1 is 8 and the number of columns is 3, we have $m = 8$ and $n = 3$. Also the length of the information bits, k is 16 . Since $\Delta_1 = 1$, then $Z = 2^{a_1} = 2^2 = 4$.

The weights of the connections between input layer 0 and layer 1 are given by

$$w_{ij}^{01} = 1 \text{ , where } i = 1, 2, \ldots, 16 \text{ and } j = 1, 2, \ldots, 8 .$$

Inputs for the layer 0, i.e. bits of the word received, are

$v_1^0 = 1, v_2^0 = 0, v_3^0 = 0, v_4^0 = 0, v_5^0 = 0, v_6^0 = 0, v_7^0 = 0, v_8^0 = 0,$

$v_9^0 = 0, v_{10}^0 = 1, v_{11}^0 = 1, v_{12}^0 = 1, v_{13}^0 = 1, v_{14}^0 = 1, v_{15}^0 = 1, v_{16}^0 = 1$

According to the proposed network, we need to find the weighted sum of these inputs. Since the weight of the each connection between layer 0 and layer 1 is 1, the weighted sum of these inputs will be the same.

Thus

$$u_i^1 = \sum_j^n w_{ji}^{01} v_j^0 = 1.1 + 1.0 + 1.0 + 1.0 + 1.0 + 1.0 + 1.0 + 1.0 + 1.0 +$$

$$1.1 + 1.1 + 1.1 + 1.1 + 1.1 + 1.1 + 1.1 = 8$$

We get $Z = 4$ and $S = \lfloor (k - u_i^1) / Z \rfloor$, $S = \lfloor (16 - 8)/4 \rfloor = 2$. We will compute the outputs of neurons in the layer 1 using the equation (5.1) as follows:

$$v_1^1 = g_1^1(u_1^1) = 0\ ,\ v_2^1 = g_2^1(u_2^1) = 0\ ,\ v_3^1 = g_3^1(u_3^1) = 0\ ,\ v_4^1 = g_4^1(u_4^1) = 0$$

$$v_5^1 = g_5^1(u_5^1) = 0\ ,\ v_6^1 = g_6^1(u_6^1) = 1\ ,\ v_7^1 = g_7^1(u_7^1) = 0\ ,\ v_8^1 = g_8^1(u_8^1) = 0\ .$$

The weights of synapse connecting layer 1 and layer 2 are :

$$w_{11}^{12} = 1, w_{21}^{12} = 1, w_{31}^{12} = 1, w_{41}^{12} = 1, w_{51}^{12} = 0, w_{61}^{12} = 0, w_{71}^{12} = 0, w_{81}^{12} = 0$$

$$w_{12}^{12} = 1, w_{22}^{12} = 1, w_{32}^{12} = 0, w_{42}^{12} = 0, w_{52}^{12} = 1, w_{62}^{12} = 1, w_{72}^{12} = 0, w_{82}^{12} = 0$$

$$w_{13}^{12} = 1, w_{23}^{12} = 0, w_{33}^{12} = 1, w_{43}^{12} = 0, w_{53}^{12} = 1, w_{63}^{12} = 0, w_{73}^{12} = 1, w_{83}^{12} = 0$$

Inputs for neurons at layer 2 are :

$$v_1^1 = 0, v_2^1 = 0, v_3^1 = 0, v_4^1 = 0, v_5^1 = 0, v_6^1 = 1, v_7^1 = 0, v_8^1 = 0,$$

The weighted sum of these inputs are:

$$u_1^2 = \sum_j^m w_{j1}^{12} v_j^1 = 1.0 + 1.0 + 1.0 + 1.0 + 0.0 + 0.1 + 0.0 + 0.0 = 0$$

$$u_2^2 = \sum_j^m w_{j2}^{12} v_j^1 = 1 \cdot 0 + 1 \cdot 0 + 0 \cdot 0 + 0 \cdot 0 + 1 \cdot 0 + 1 \cdot 1 + 0 \cdot 0 + 0 \cdot 0 = 1$$

$$u_3^2 = \sum_j^m w_{j3}^{12} v_j^1 = 1 \cdot 0 + 0 \cdot 0 + 1 \cdot 0 + 0 \cdot 0 + 1 \cdot 0 + 0 \cdot 1 + 1 \cdot 0 + 0 \cdot 0 = 0$$

Since we have defined the threshold logic function as an activation function for each neuron, the outputs of neurons in the layer 2 are :

$$v_1^2 = g_2^2(u_1^2) = 0$$

$$v_2^2 = g_2^2(u_2^2) = 1$$

$$v_3^2 = g_2^2(u_3^2) = 0$$

Thus for the given information bits X ; 1000000001111111 ,

the check bits R_{1,i_1} are 010 .

We can also find the check bits R_{2,i_2} by using matrix M_2 . In this case, since the number of rows in the matrix M_2 is 3 and the number of columns is 4, we have $m = 3$ and $n = 4$. Also the length of the information bits, k is 16 . Since $\Delta_1 = 2 > 1$, then $Z = d - t + 1 + 2.\Delta_1 = 5 - 2 + 1 + 2 = 6$.

The weights of the connections between input layer 0 and layer 1 are given by

$$w_{ij}^{01} = 1 \text{ , where } i = 1, 2, \ldots, 16 \text{ and } j = 1, 2, \ldots, 3 .$$

Inputs for the layer 0, i.e. bits of the word received, are

$$v_1^0 = 1,\ v_2^0 = 0,\ v_3^0 = 0,\ v_4^0 = 0,\ v_5^0 = 0,\ v_6^0 = 0,\ v_7^0 = 0,\ v_8^0 = 0,$$

$$v_9^0 = 0,\ v_{10}^0 = 1,\ v_{11}^0 = 1,\ v_{12}^0 = 1,\ v_{13}^0 = 1,\ v_{14}^0 = 1,\ v_{15}^0 = 1,\ v_{16}^0 = 1$$

According to the network, we need to find the weighted sum of these inputs. Since the weight of the each connection between layer 0 and layer 1 is 1, the weighted sum of these inputs will be the same.

Thus

$$u_t^1 = \sum_j^n w_{jt}^{01} v_j^0 = 1.1 + 1.0 + 1.0 + 1.0 + 1.0 + 1.0 + 1.0 + 1.0 + 1.0 +$$

$$1.1 + 1.1 + 1.1 + 1.1 + 1.1 + 1.1 + 1.1 = 8$$

We get $Z = 4$ and $S = \lfloor (k - u_t^1) / Z \rfloor$, $S = \lfloor (16 - 8)/6 \rfloor = 1$, then we will compute the outputs of neurons in the layer 1 using the equation (5.1) as follows: follow:

$$v_1^1 = g_1^1(u_1^1) = 0$$

$$v_2^1 = g_2^1(u_2^1) = 1$$

$$v_3^1 = g_3^1(u_3^1) = 0$$

The weights of synapse connecting layer 1 and layer 2 are :

$$w_{11}^{12} = 1, w_{21}^{12} = 0, w_{31}^{12} = 0$$

$$w_{12}^{12} = 1, w_{22}^{12} = 0, w_{32}^{12} = 0$$

$$w_{13}^{12} = 1, w_{23}^{12} = 1, w_{33}^{12} = 0$$

$$w_{14}^{12} = 1, w_{24}^{12} = 1, w_{34}^{12} = 0$$

Inputs for neurons at layer 2 are :

$$v_1^1 = 0, v_2^1 = 1, v_3^1 = 0$$

The weighted sum of these inputs are:

$$u_1^2 = \sum_j^m w_{j1}^{12} v_j^1 = 1 \,.\, 0 + 0 \,.\, 1 + 0 \,.\, 0 \; = 0$$

$$u_2^2 = \sum_j^m w_{j2}^{12} v_j^1 = 1 \,.\, 0 + 0 \,.\, 1 + 0 \,.\, 0 \; = 0$$

$$u_3^2 = \sum_j^m w_{j3}^{12} v_j^1 = 1 \,.\, 0 + 1 \,.\, 1 + 0 \,.\, 0 \; = 1$$

$$u_4^2 = \sum_j^m w_{j4}^{12} v_j^1 = 1 \,.\, 0 + 1 \,.\, 1 + 0 \,.\, 0 \; = 1$$

Since we have defined the threshold logic function (5.2) as an activation function for each neuron, the outputs of neurons in the layer 2 are :

$$v_1^2 = g_2^2(u_1^2) = 0$$

$$v_2^2 = g_2^2(u_2^2) = 0$$

$$v_3^2 = g_2^2(u_3^2) = 1$$

$$v_4^2 = g_2^2(u_4^2) = 1$$

Thus for the given information bits X ; 1000000001111111, we get the check bits R_{2,i_2}, 0011 . Thus the required codeword is

100 0000001111111 010 0011

X R_{1,i_1} R_{2,i_2}

which can be checked using Table 5.3 2-EC/5-ED/AUED Code II.

In this way, we can construct the required t-EC/d-ED/AUED code which contains codewords of the form $XR_{1,i_1}R_{2,i_2}$.

5.3.3 A Scheme for Error Detection/Correction

We have described error correcting network for linear codes in Chapter 3, Section 3.3. According to the Section 5.2.4 and Section 5.2.4.1, detecting and correcting algorithm is almost the same as the algorithm in Section 3.3. So we will use the same network structure of Section 3.3. In this case we will show detecting and correcting a single error. The remaining part will be left for future research. We will demonstrate the network shown in Section 3.3 using the t-EC/d-Ed/AUED codes in the next section.

5.3.3.1 Example

As we mentioned in Section 5.2.2, we must have a systematic t-EC/d-ED parity check code F which has a parity check matrix H.

$$H = \begin{bmatrix}
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}.$$

Let the word 110 0000001111111 00011000 be the information bits of the received word. We assume that this received words contains some error.

In this problem, since the number of row in the check matrix is 13 and the length of the codeword is 16, we have $M = 13$ and $N = 16$. According to Section 3.3, the weights of the synapse connecting between input layer 0 and layer 1 are defined as

$$w_{ij}^{01} = h_{ji}, 1 \le i \le N \text{ and } 1 \le j \le M,$$

where w_{ij}^{01} is the weight of the *ith* neuron of the input layer with the *jth* neuron of layer 1, and h_{ij} is an element of row *ith* and column *jth* of matrix H.

Inputs for the layer 1, i.e. bits of the word received, are

$v_1^0 = 1, v_2^0 = 1, v_3^0 = 0\ v_4^0 = 0, v_5^0 = 0, v_6^0 = 0\ v_7^0 = 0, v_8^0 = 0$

$v_9^0 = 0\ v_{10}^0 = 1, v_{11}^0 = 1, v_{12}^0 = 1\ v_{13}^0 = 1, v_{14}^0 = 1, v_{15}^0 = 1\ v_{16}^0 = 1$

According to the proposed network, we need to find the weighted sum of these inputs as follows:

$$u_1^1 = \sum_j^N w_{j1}^{01} v_j^0 = 0, u_2^1 = \sum_j^N w_{j2}^{01} v_j^0 = 0, u_3^1 = \sum_j^N w_{j3}^{01} v_j^0 = 0, u_4^1 = \sum_j^N w_{j4}^{01} v_j^0 = 1$$

$$u_5^1 = \sum_j^N w_{j5}^{01} v_j^0 = 1, u_6^1 = \sum_j^N w_{j6}^{01} v_j^0 = 1, u_7^1 = \sum_j^N w_{j7}^{01} v_j^0 = 3, u_8^1 = \sum_j^N w_{j8}^{01} v_j^0 = 3$$

$$u_9^1 = \sum_j^N w_{j9}^{01} v_j^0 = 3, u_{10}^1 = \sum_j^N w_{j10}^{01} v_j^0 = 2, u_{11}^1 = \sum_j^N w_{j11}^{01} v_j^0 = 2, u_{12}^1 = \sum_j^N w_{j12}^{01} v_j^0 = 2$$

$$u_{13}^1 = \sum_j^N w_{j13}^{01} v_j^0 = 3$$

After applying the activation function (3.1) shown in Chapter 3, the outputs of neurons in the layer 1 are :

$$v_1^1 = g^1(u_1^1) = -1\ ,\ v_2^1 = g^1(u_2^1) = -1\ ,\ v_3^1 = g^1(u_3^1) = -1\ ,\ v_4^1 = g^1(u_4^1) = 1$$

$$v_5^1 = g^1(u_5^1) = 1\ ,\ v_6^1 = g^1(u_6^1) = 1\ ,\ v_7^1 = g^1(u_7^1) = 1\ ,\ v_8^1 = g^1(u_8^1) = 1$$

$$v_9^1 = g^1(u_9^1) = 1\ ,\ v_{10}^1 = g^1(u_{10}^1) = -1\ ,\ v_{11}^1 = g^1(u_{11}^1) = -1\ ,\ v_{12}^1 = g^1(u_{12}^1) = -1$$

$$v_{13}^1 = g^1(u_{13}^1) = -1$$

According to Section 3.3, the weights of synapse connecting layer 1 and layer 2 are defined as follow: For $1 \le i \le M, 1 \le j \le N$

$$w_{ij}^{12} = \begin{cases} 1 & \text{if } h_{ij=1} \\ -1 & \textit{otherwise} \end{cases}$$

Inputs for neurons at layer 2 are :

$$v_1^1 = -1,\ v_2^1 = -1,\ v_3^1 = -1,\ v_4^1 = 1$$

$$v_5^1 = 1,\ v_6^1 = 1,\ v_7^1 = 1,\ v_8^1 = 1$$

$$v_9^1 = 1,\ v_{10}^1 = -1,\ v_{11}^1 = -1,\ v_{12}^1 = -1$$

$$v_{13}^1 = 1$$

The weighted sum of these inputs are:

$$u_1^2 = \sum_j^M w_{j1}^{12} v_j^1 = 1,\ u_2^2 = \sum_j^M w_{j2}^{12} v_j^1 = 13,\ u_3^2 = \sum_j^M w_{j3}^{12} v_j^1 = 1,\ u_4^2 = \sum_j^M w_{j4}^{12} v_j^1 = -3$$

$$u_5^2 = \sum_j^M w_{j5}^{12} v_j^1 = -3,\ u_6^2 = \sum_j^M w_{j6}^{12} v_j^1 = -3,\ u_7^2 = \sum_j^M w_{j7}^{12} v_j^1 = 1,\ u_8^2 = \sum_j^M w_{j8}^{12} v_j^1 = 1$$

$$u_9^2 = \sum_j^M w_{j9}^{12} v_j^1 = 1,\ u_{10}^2 = \sum_j^M w_{j10}^{12} v_j^1 = 1,\ u_{11}^2 = \sum_j^M w_{j11}^{12} v_j^1 = 1,\ u_{12}^2 = \sum_j^M w_{j12}^{12} v_j^1 = 1$$

$$u_{13}^2 = \sum_j^M w_{j13}^{12} v_j^1 = -3,\ u_{14}^2 = \sum_j^M w_{j14}^{12} v_j^1 = -3,\ u_{15}^2 = \sum_j^M w_{j15}^{12} v_j^1 = -3,\ u_{16}^2 = \sum_j^M w_{j16}^{12} v_j^1 = 1$$

Here we use the activation function (3.2) shown Chapter 3 and threshold θ = M - 1/2 = 13 - 1/2 = 12.5 and , the outputs of neurons in the layer 2 are :

$$v_1^2 = g^2(u_1^2) = 0,\ v_2^2 = g^2(u_2^2) = 1,\ v_3^2 = g^2(u_3^2) = 0,\ v_4^2 = g^2(u_4^2) = 0$$

$$v_5^2 = g^2(u_5^2) = 0,\ v_6^2 = g^2(u_6^2) = 0,\ v_7^2 = g^2(u_7^2) = 0,\ v_8^2 = g^2(u_8^2) = 0$$

$$v_9^2 = g^2(u_9^2) = 0,\ v_{10}^2 = g^2(u_{10}^2) = 0,\ v_{11}^2 = g^2(u_{11}^2) = 0,\ v_{12}^2 = g^2(u_{12}^2) = 0$$

$$v_{13}^2 = g^2(u_{13}^2) = 0,\ v_{14}^2 = g^2(u_{14}^2) = 0,\ v_{15}^2 = g^2(u_{15}^2) = 0,\ v_{16}^2 = g^2(u_{16}^2) = 0$$

After the first phase, we have detected that second position of the given word is in error. In the second phase, we use the XOR network shown in Section 3.3 with the inputs of the corresponding bit positions of the output of phase 1 (0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0) and the received word (1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1). Then according to the error correcting phase shown in Section 3.3.2, the XOR network produces the correct codeword (1 0 0 0 0 0 0 0 1 1 1 1 1 1 1).

Chapter 6

Conclusions

6.1 Summary

As mentioned in the previous chapters, many researchers have developed error correcting codes in many different ways. But according to our knowledge, no one has discussed how to implement error correcting codes in the framework of neural computing. This research might be the first one which concentrates on developing error correcting codes using neural networks. Since neural networks have been studied for many years for the purpose of achieving human-like performance in the fields of speech and image recognition and these models are composed of many nonlinear computational elements operating in parallel and arranged in patterns reminiscent of biological neural nets, we hope that our error correcting paradigm can be applied in neural computing field to gain the benefits of parallel processing. So we propose to design and construct a neural network as a new approach for error detection/correction.

We have presented some specific issues in this research. The application of modeling of neural networks might be complemented by deep insight into how to embed algorithm for correcting errors in linear codes. Thus we presented an error-correcting algorithm in the framework of neural networks in Chapter 3. We considered the problem of detection and correction of a linear code which is assumed to have at most one error. Follow-up studies were also made in Chapter 4 and 5 to gain better insight into the general problem of error-correcting and neurocomputing formalism for modeling of error correcting paradigms with neural circuits. In Chapter 4, we constructed neural networks for detecting double and triple errors and described a heuristic algorithm for

detecting multiple errors in systematic unidirectional codes. In Chapter 5, we presented construction of similar networks for t-error correcting/d-error detecting and all unidirectional error detecting codes and illustrative examples were also provided.

6.2 Benefits

The quest for an efficient computational approach to neural connectivity problems has undergone a significant evolution in the last few years. The current best systems are far from equaling human performance especially when a program of instructions is executed sequentially as in a von Neuman computer. On the other hand, neural nets models are potential candidates for parallel processing since they explore many competing hypotheses simultaneously using massively parallel nets composed of many computational elements connected by links with variable weights. Neural nets models have many processors, each executing a very simple program, instead of the conventional situation where one or just a few processors execute very complicated programs. In contrast to the robustness of a neural network, an ordinary sequential computation may easily be ruined by a single bit error. Nevertheless, the brain can do very fast processing for tasks like vision, motor control, and decisions on the basis of incomplete and noisy data, tasks that are far beyond the capacity of a current supercomputer [14]. It seems possible only because billions of neurons operate simultaneously.

As we constructed a network for single error detection and correction in linear codes, it may be applied in communications, control and computing. Then we presented an error-detecting paradigm in the framework of neural networks. We considered the problem of error detection of systematic unidirectional codes which is

circuit using a tree-type r bit 2's complement adder to calculate the value of k_0 mod m, where m is some appropriate integer. In this circuit, the longer we have information bits, the higher the level of tree has to be constructed. Since our proposed network has only two levels regardless of the length of information part, it can generate the check bits faster than the previous circuit. The proposed network can also be generalized according to the construction of codes by locating the number of neurons in order to the number of check bits and finding the appropriate activation functions of each neurons at layer 1, as shown in Section 4.6. We also described models of the code construction, detection and correction of t-EC/d-ED/AUED codes which are more general codes in error correcting paradigm. In practice, the proposed networks might gain better insight into error detecting/correcting paradigm concerning with faster time complexity.

Throughout our research, we designed models and tested with the simulation programs. The results of simulation runs for this work convinced us of the correctness, convergence and high speed computation of our approach. We may also continue work on the emerging area of dynamic systems in the context of neural network. The advantage of this research can be applied for the real time error detection and correction and parallel computation in the context of information flow in pipelied implementation.

6.3 The Principle Contributions of the Dissertation

Based on the foregoing discussions we summarize the principle contributions of this dissertation and these are as follows:

- New algorithmic structure with real time cost reduction for single error detection and correction in linear codes is presented.

- A generalization of the above algorithm for double error-detecting network model is presented and it has a better performance in terms of data structure and speed than the B.Bose and D.J. Lin's adder circuit[8].

- We have also shown that the triple error-detecting network model is a simple extension of the double error-detecting network model.

- One of the distinguishing features of our models is in a heuristic algorithm which is able to construct an efficient new model of neural work for detecting multiple errors in systematic unidirectional codes.

- Our new algorithm for t-EC/d-ED/AUED(t-Error Correcting/d-Error Detecting/ All Unidirectional Error Detecting)code construction is a real motivation for studying error correcting codes using ANNs.

- Finally, the last feature of our technique is in its novel parallel data structure for error correcting paradigm in the context of neural network.

6.4 Future Research

Based on our research, there are some interesting problems or questions still remain to be solved.

- It would be interesting to design a network to detect and correct multiple errors in linear codes. Our approach may be applied and the result may be similar.

- Networks for double and triple error-detecting codes and a heuristic algorithm of network construction for systematic unidirectional multiple error-detecting codes were discussed. Detailed investigations could lead to improve in the heuristic algorithm.

- Multiple error detection in t-EC/d-ED/AUED(t-Error Correcting/d-Error Detecting/ All Unidirectional Detecting) codes needs to be tested with our proposed network. It might be necessary to improve the design of the network in terms of activations and/or weights.

Bibliography

[1]. Norman L. Biggs "Discrete Mathematics", Revised edition 1989, Oxford University Press.

[2]. M. Blaum and H. V. Tilborg, "On t-Error Correcting/All Unidirectional Error Detecting Codes", *IEEE Transactions on Computers* vol 38, pp-1493-1501, Nov. 1989.

[3]. B. Bose, "Theory and design of unidirectional error codes," Ph.D. Dissertation, Dep. Computer Sci. Eng., Southern Methodist Univ., Dallas, TX. May 1980.

[4]. B. Bose, "On systematic SEC/MUED code," in *Proceeding FTCS.* vol. 11, June 1981, pp. 265-267.

[5]. B. Bose and T. R. Rao, " Theory of Unidirectional Error Correcting/ Detecting Codes", *IEEE Transactions on Computers* vol c-31, pp-521-530. June, 1982.

[6]. B. Bose and D. K. Pradhan, "Optimal Unidirectional Error Detecting/ Correcting Codes", *IEEE Transactions on Computers* vol c-31, pp 564-568, June, 1982.

[7]. B. Bose, "On Unordered Codes", *IEEE Transactions on Computers* vol 40, pp-125-131, Feb. 1991.

[8]. B. Bose and D. J. Lin, "Systematic Unidirectional Error-Detecting Codes" *IEEE Transactions on Computers* vol 34, pp-1026-1032, Nov. 1985.

[9]. R. W. Cook, W. H. Sisson, T. F. Storey, and W. W. Toy, "Design of a self-checking microprogram control," *IEEE Transactions on Computers,* vol. C-22, pp. 255-262, Mar. 1973.

[10]. D. J. Evans, M. Adamopoulos, S.Kortesis, and K. Tsouros "Searching sets of properties with neural networks." Parallel Computing 16(1990) 279-285 North-Holland.

[11]. J.A Feldman and D.H. Ballard, "Connectionist Models and Their Properties" *Cognitive Science, Vol 6,* 205-254, 1982.

[12]. S. Grossberg, *The Adaptive Brain I: Cognition, Learning, Reinforcement, and Rhythm, and The Adaptive Brain II: Vision, Speech, Language, and Motor Control,* Elsevier/North-Holland, Amsterdam(1986).

[13]. D.O. Hebb, *The Organization of Behavior*, John Wiley & Sons, New York (1949).

[14]. John Hertz, Anders Kaogh, and Richard G. Palmar, "Introduction to the theory of neural computing." 1991, Addison Welsley Publishing Company.

[15]. J.J.Hopfield, "Neurals Networks and Physical Systems with Emergent Collective Computational Abilities," *Proc. Natl. Acad. Sci.* U.S.A, Vol. 79, 2554-2558, April 1982.

[16]. J.J.Hopfield, "Neurals with Graded Response Have Collective Computational Properties Like Those of Two-State Neurons," *Proc. Natl. Acad. Sci.* U.S.A, Vol. 81, 3088-3092, May 1984.

[17]. J.J. Hopfield, and D.W. Tank, "Computing with Neural Circuits: A Model," *Science*, Vol 233, 625-633, August 1986.

[18]. M. M. Htay, S. S. Iyengar, and S. Q. Zheng, " Correcting Errors in Linear Codes with Neural Network", Technical Report, Department of Computer Science, Nov. 1991.

[19]. S. S. Iyengar "Computer Modeling of Biological System", CRC press, 1982.

[20]. D. E. Knuth, "Efficient Balance Codes", *IEEE Transactions on Computers* vol IT-32, pp-51-53, Jan. 1986.

[21]. D. J. Lin and B. Bose, "Theory and design of t-error correcting and d(d>t)-unidirectional error detecting(t-EC d-UED) codes," *IEEE Transactions on Computers*, vol 37, pp. 433-439, Apr. 1988.

[22]. R. P. Lippmann, "Introduction to computing with neural nets." *IEEE Transactions on Acoustics, Speech and Signal Processing* (April 1987) pp-4-22.

[23]. J. H. van Litt, "Introduction to Coding", 1982, Springer-Verlag, New York Inc.

[24]. D. Nikolos, N. Gaitanis, and G. Philokyprou, "Systematic t-Error Correcting/All Unidirectional Error Detecting Codes", *IEEE Transactions Computer* vol c-35, pp-394-402, May, 1986.

[25]. D. Nikolos, " Theory and Design of t-Error Correcting/ d-Error Detecting (d > t) and All Unidirectional Error Detecting Codes", *IEEE Transactions on Computers* vol 40, pp-132-142, Feb. 1991.

[26]. B. Parhami and A. Avizienis, " Detection of storage errors in mass memories using low-cost arithmetic codes," *IEEE Transactions on Computers*, vol C-27, pp 302-308, Apr. 1978.

[27]. W. Wesley Peterson, " Error Correcting Codes ", The M.I.T. Press, 1961.

[28]. Vera Pless, " Introduction to The Theory of Error-Correcting Codes", Wiley-Interscience Series in Discrete Mathematics, John Wiley & Son, Inc 1982.

[29]. D.K. Pradhan and S.M. Reddy, "Fault-tolerant failsafe logic networks," in *Proceeding IEEE COMPCON*, San Francisco, CA, Mar. 1977, p.363.

[30]. D. K. Pradhan, "A New Class of Error-Correcting/Detecting Codes for Fault-Tolerant Computer Applications", *IEEE Transactions on Computers* vol c-29, pp. 471-481. June, 1980.

[31]. T. R. N. Rao and E. Fujiwara, "Error-Control Coding For Computer System" Prentice Hall, 1989.

[32]. D. L. Tao, C. R. P. Hartmann, and P.K. Lala, "An efficient class of unidirectional error detecting/correcting codes," *IEEE Transactions, Computer,* vol. 37, pp 879-882, July 1988.

[33]. D. Tasar and V. Tasar, " A study of intermittent faults in digital computers", in FIProc. AFIPS Conference, 1977, pp 807-811.

[34]. J. F. Wakerly, "Detection of unidirectional multiple errors using low cost arithmetic codes," *IEEE Transactions on Computers*," vol C-24 vol C-24, pp 210-212, Feb. 1975.

[35]. J. H. Weber, C. de Vroedt, and Dick E. Boekee, "Bounds and Constructions for Binary Codes of Length Less than 24 and Asymmetric Distance Less than 6", *IEEE Transactions on Information Theory*, Vol 34, No. 5, Sept. 1988.

[36]. ---, "systematic t-error correcting/all unidirectional detecting codes," *IEEE Transactions on Computers*, vol C-35, pp 394-402, May 1986.

VITA

Maung Maung Htay was born in Kyaiklat, Union of Myanmar (formerly Burma). He received his B.S. and M.S. degrees in Mathematics from the University of Rangoon in 1971 and 1979, respectively. He also obtained his M.S. degree in Computer Science from the University of London in 1976. He joined the Ph.D. program in the Department of Computer Science at Louisiana State University, Baton Rouge, Louisiana in August, 1988.

His current research interests include parallel processing, artificial intelligence, fault tolerance, error correcting codes, and neural networks.

DOCTORAL EXAMINATION AND DISSERTATION REPORT

Candidate: Maung Maung Htay

Major Field: Computer Science

Title of Dissertation: A Computational Framework For Efficient Correcting Codes Using An Artificial Neural Network Paradigm

Approved:

Major Professor and Chairman

Dean of the Graduate School

EXAMINING COMMITTEE:

Date of Examination:

July 15, 1992